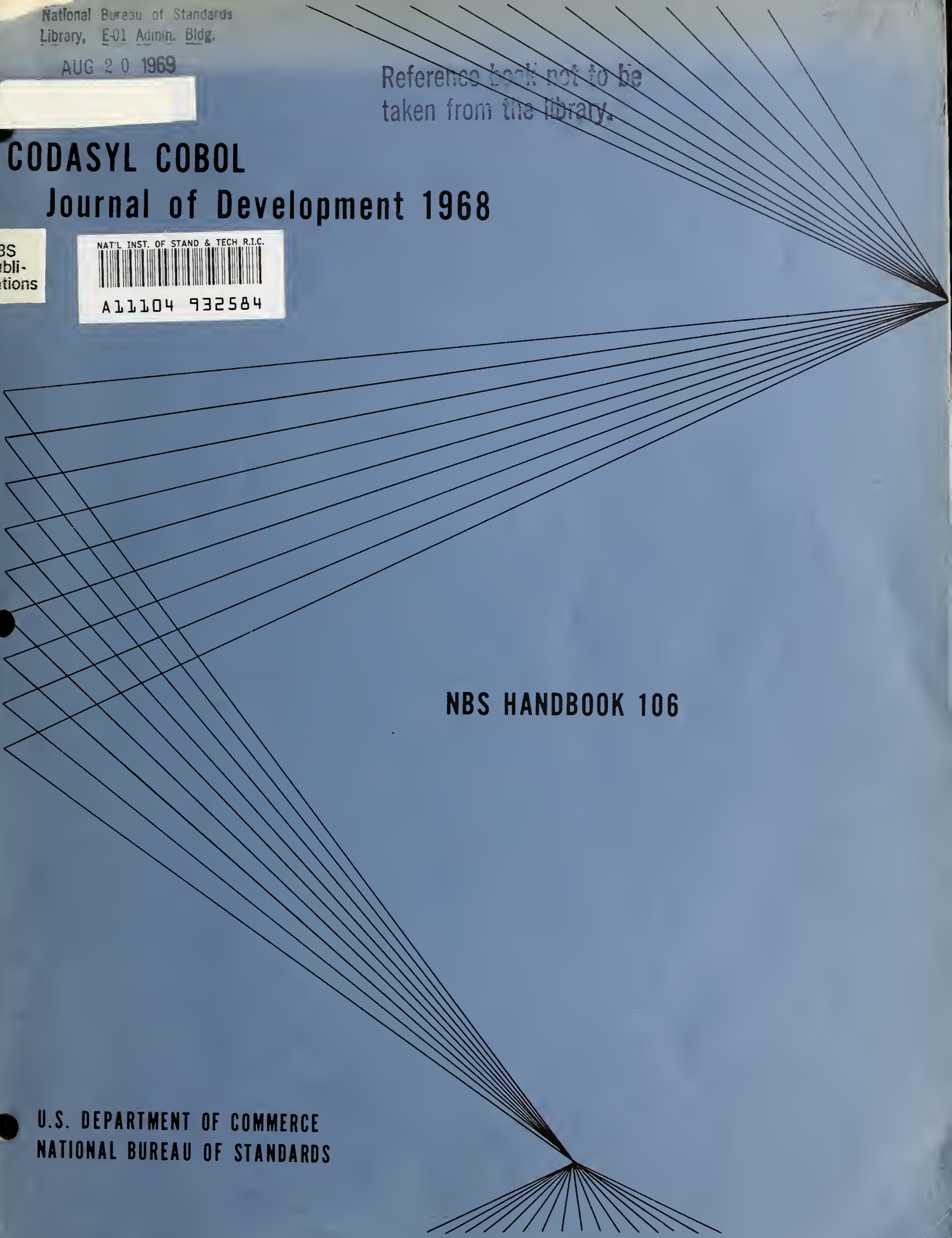
National Bureau of Standards
Library, E-01 Admin. Bldg.
AUG 20 1969

Reference book not to be
taken from the library.

CODASYL COBOL
Journal of Development 1968

NBS
Publi-
cations

NAT'L INST. OF STAND & TECH R.I.C.
A11104 932584



NBS HANDBOOK 106

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

UNITED STATES DEPARTMENT OF COMMERCE • Maurice H. Stans, *Secretary*
NATIONAL BUREAU OF STANDARDS • A. V. Astin, *Director*

CODASYL COBOL
Journal of Development 1968

t.
U.S. National Bureau of Standards, Handbook 106,

Nat. Bur. Stand. (U.S.), Handb. 106, 344 pages (July 1969)
CODEN: NBSHA

Issued July 1969

For sale by the Superintendent of Documents, U.S. Government Printing Office
Washington, D.C. 20402 - Price $2.75

NATIONAL BUREAU OF STANDARDS
JAN 12 1970
Not acc. Ref.
QC1
.U51
No 106
1969
Copy 1

Abstract

This document is a report to the COBOL community from the Conference on Data Systems Languages (CODASYL) Common Business Oriented Language (COBOL) Programming Language Committee. It is an official report documenting the development activities of CODASYL through July 1968.

Key words: COBOL; CODASYL; journal.

History of COBOL Specification Documents

COBOL — 60, published 1960
COBOL — 61, published 1961
COBOL — 61 Extended, published 1963
COBOL — 65, published 1965
COBOL — Journal of Development — 1968, published 1968

Library of Congress Catalog Card No.: 73-601243

Foreword

Under Public Law 89-306 (Brooks Bill) the Secretary of Commerce was given important responsibilities for improving the procurement, utilization, and management of computers and related information systems in the Federal Government. To carry out the Secretary's responsibilities under the Brooks Bill, the NBS Center for Computer Sciences and Technology provides leadership and coordination for Government efforts in the development of voluntary commercial information processing standards.

A major problem in the use of electronic data processing equipment lies in the inability to state the data processing application in such a way that computer programs are developed and maintained with a minimum of time and programming effort. A common business-oriented computer language, independent of any make or model of computer, would do much to solve this problem.

Since 1959, the Conference on Data Systems Languages (CODASYL) has been active in the development, specification, and maintenance of a COmmon Business Oriented Language (COBOL). The current activity within CODASYL on the development of COBOL is being conducted by the Programming Language Committee, composed of voluntary representatives from computer manufacturers and users in industry and the Federal Government.

The present publication represents a report to the COBOL community from the CODASYL Programming Language Committee on the development of COBOL through July 1968. The National Bureau of Standards is pleased to have the opportunity to make this information available through publication as an NBS Handbook.

A. V. ASTIN, *Director*

PREFACE

This document is a report to the COBOL community from the CODASYL COBOL Programming Language Committee. It is an official report documenting the development activities of CODASYL through July, 1968 and is subject to change, extension and further development. Neither this document nor any of its precursors (i.e. COBOL-60, COBOL-61, COBOL-61 Extended, COBOL-65) are to be construed as official standards. Standardization of COBOL in the United States is in the purview of USASI Committee X3 and it rather than CODASYL is responsible for the preparation and submission for approval of a COBOL standard.

COBOL, Journal of Development, is composed of three sections as follows:

Section I.	History of CODASYL COBOL Development
Section II.	Philosophy of COBOL Use
Section III.	COBOL Language Specifications

It is intended that the Journal of Development be published no more often than once each year and no less often than every three years.

The objectives for the different sections are as follows:

Section I.	As a chronological record, this section is a brief history of the CODASYL organization and of the development of the COBOL Language definition from 1959 through the specification and publication of the present document. Participating companies during this period are noted as well as the official documents that have been published by the CODASYL Executive Committee.
Section II.	In the design and use of the COBOL language specifications, a philosophy of language use exists from both an implementor and a user point of view. This Section is included as an expression of language guidelines to the user of the language. To aid the depth understanding of both the

implementor and user, guidelines include background discussion of extended language features whose data processing concepts are not always evident. Guidelines also include objectives that the COBOL designers considered in the need for a common language and discuss the practical experience that is known today in relation to the original objectives.

Section III. This self-contained Section is a complete, formal, semantic description of the COBOL language specification. The need for a thorough language definition, without philosophical treatment of design criteria, is the objective in the presentation of the language.

ACKNOWLEDGMENT

The COBOL Journal of Development is the product of the CODASYL Programming Language Committee. This effort has been aided significantly by the contributions of the European Computer Manufacturer's Association (ECMA), the International Organization for Standardization (ISO), the Japanese COBOL Standards Committee, the United States of America Standards Institute (USASI), and by other interested organizations and individuals. The journal was developed by the Programming Language Publication Subcommittee and reflects the official CODASYL COBOL language as of July, 1968.

The maintenance of COBOL is the function of the Programming Language Committee of CODASYL (COnference on DAta SYstems Languages). Proposed changes to the language specifications are introduced by written proposal. Although no specific proposal format is mandatory the following guidelines are applied to proposal content:

1. Proposals made to specific points must cite all specific references.
2. Proposals of a general nature should cite at least specific instances.
3. Sufficient justification and motivation should be contained in the proposal to point out what the problem appears to be and why this proposal is a solution.
4. The proposal should include recommended specification changes with specific references where necessary.

Additional information concerning the procedures for proposing changes or suggested changes should be directed to the Programming Language Committee:

Chairman, Programming Language Committee
Box 124
Monroeville, Pennsylvania 15146

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment.

> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

SECTION I: HISTORY OF CODASYL COBOL DEVELOPMENT

TABLE OF CONTENTS

SECTION I: HISTORY OF CODASYL COBOL DEVELOPMENT

Page

CHAPTER 1. BACKGROUND and INTRODUCTION I-1-1

1.1 Objectives I-1-1
1.2 Establishment of CODASYL I-1-2

CHAPTER 2. DEVELOPMENT of COBOL I-2-1

2.1 Organization of COBOL Effort I-2-1
2.2 Evolution of COBOL I-2-6
2.3 Standardization I-2-9
2.4 Future Developments I-2-9

CHAPTER 1
BACKGROUND and INTRODUCTION

1.1 OBJECTIVES

On May 28 and 29, 1959 a meeting was held in the Pentagon for the purpose of considering both the desirability and the feasibility of establishing a common language for the programming of electronic computers for business-type applications. Representatives from users, both in private industry and in government, computer manufacturers, and other interested parties were present. The group agreed that the project should be undertaken. The following general objective was stated:

> The current experience of users of electronic data processing equipment indicates that a major problem in the efficient utilization of such equipment lies in the inability to state the data processing application in such a way that computer programs are developed and maintained with a minimum of time and programming effort.
>
> A COmmon Business Oriented Language, independent of any make or model of computer, open ended, and stated in both an English notation and a narrative form, would do much to solve or to ameliorate this problem. Such a language would also simplify and speed up the solution of the related problem of training personnel in the design of data processing systems and in the development of computer programs for such systems.
>
> In general, the development of a common language would serve to benefit the user in the following situations.
>
>> In developing data processing systems for existing computers, it is important that these systems be capable of processing on future, more powerful computers of any manufacturer, with a minimum of conversion costs.
>>
>> Full documentation of present systems in a form conducive to making changes and additions with minimum expenditures of time and cost is necessary in order to effectively meet the rapidly changing and expanding requirements of management.
>>
>> The need to produce a large number of computer programs in a short period of time often requires the augmentation of programming staffs by the addition of relatively inexperienced programmers.

1.2 ESTABLISHMENT OF CODASYL

The concept of the three committees was agreed upon and Short Range, Intermediate Range, and Long Range committees were established. The Short Range Committee was given the task of developing an immediate language and was instructed to take the best of three existing language-compiler systems, FLOWMATIC, AIMACO, and Commercial Translator, and to produce a language superior to any of these. The COnference on DAta SYstems Languages (CODASYL) developed out of that meeting. The Short Range Committee eventually became the official COBOL branch of CODASYL and the Intermediate and the Long Range Committees evolved into the Systems and Language Structures Committees.

CHAPTER 2

DEVELOPMENT OF COBOL

2.1 ORGANIZATION OF COBOL EFFORT

The original COBOL organization was the Short Range Committee of CODASYL. By September, 1959, this committee had specified a language which they considered superior to existing language-compiler systems. This language specification was further modified and by December, 1959, COBOL existed as a language that was not identified with any manufacturer and therefore presented advantages for both government and private industry users.

2.1.1 INITIAL ORGANIZATION

The product of phase I of COBOL development was a report published in April of 1960 by the Government Printing Office entitled "COBOL--A Report to the Conference on Data Systems Languages, including Initial Specifications for a Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers". The language described in this report has since become known as COBOL-60.

The organizations participating in the original COBOL development were:

Air Materiel Command, United States Air Force
Bureau of Standards, Department of Commerce
David Taylor Model Basin, Bureau of Ships, U.S. Navy
EDP Division, Minneapolis-Honeywell Regulator Co.
Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products Incorporated
Univac Division of Sperry Rand Corporation

2.1.2 THE COBOL MAINTENANCE COMMITTEE

The Executive Committee recognized that the task of defining COBOL was a continuing one and that the language had to be maintained and improved. To this end, the COBOL Maintenance Committee was created and charged with the task of answering questions arising from users and implementors of the language and making definitive modifications, including additions, clarifications, and changes to the language.

The Maintenance Committee was comprised of a Users Group and a Manufacturers Group. These groups met together but voted on proposals separately.

In order to devote concentrated attention to publishing a revised and updated "COBOL-60", the Executive Committee created a Special Task Group. The product of this task group was the COBOL-61 manual, which was published by the Government Printing Office in mid-1961.

The next official COBOL publication was also the product of the Maintenance Committee and was called COBOL-61 Extended; released in mid-1963.

Organizations participating in the Maintenance Committee or the Special Task Group were:

Air Materiel Command, United States Air Force
Allstate Insurance Company
Bendix Corporation, Computer Division
The Boeing Company
Burroughs Corporation
Chase Manhattan Bank
Control Data Corporation
David Taylor Model Basin, Bureau of Ships, U.S. Navy
DuPont Company
General Electric Company
General Motors Corporation
International Business Machines Corporation
Lockheed Aircraft Corporation
Minneapolis-Honeywell Regulator Company
National Cash Register Company
Owens-Illinois, Incorporated
Philco Corporation
Radio Corporation of America
Royal McBee Corporation
Space Technology Laboratories, Incorporated
Southern Railway Company
Standard Oil Company (N.J.)
Sylvania Electric Products, Incorporated
Univac Division of Sperry Rand Corporation
United States Steel Corporation
Westinghouse Electric Corporation

The membership of the various Subcommittees of the COBOL Maintenance Committee was composed of members of both the Users and the Manufacturers groups.

2.1.3 THE COBOL COMMITTEE

In January, 1964, the COBOL Maintenance Committee was reorganized to provide a true industry group and to broaden its scope of activities. This organization is shown in Figure 1.

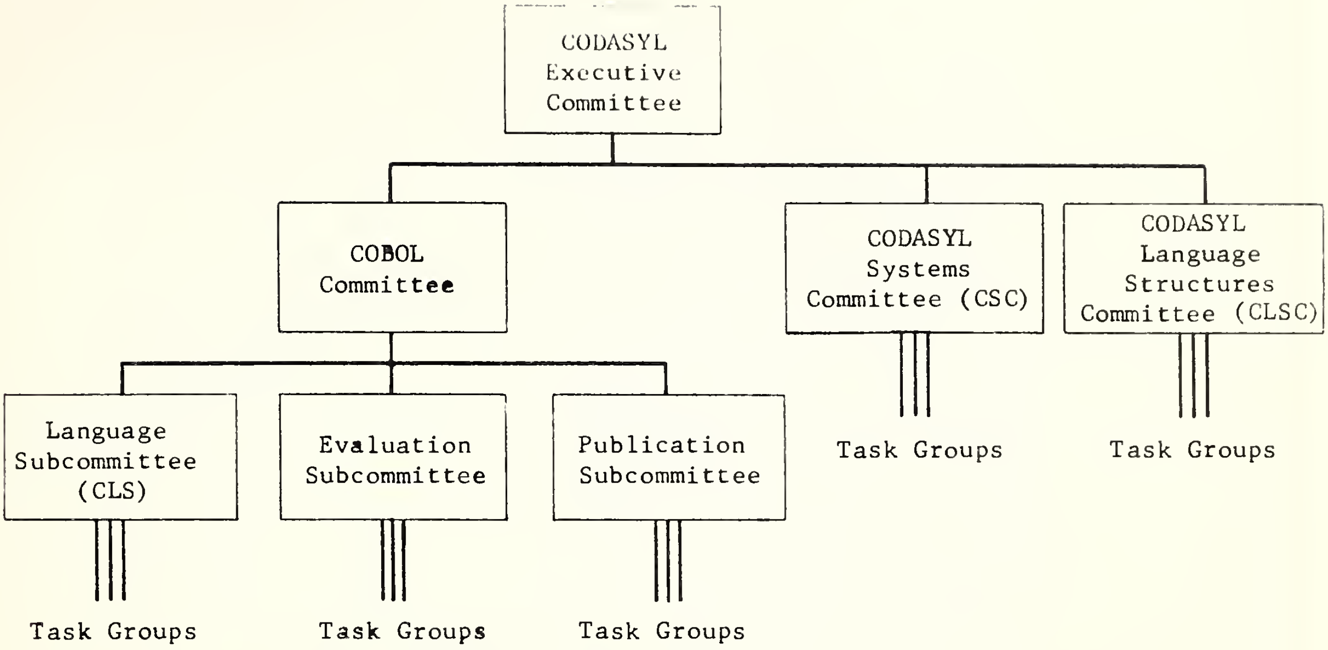

CODASYL Executive Committee
COBOL Committee
CODASYL Systems Committee (CSC)
CODASYL Language Structures Committee (CLSC)
Language Subcommittee (CLS)
Evaluation Subcommittee
Publication Subcommittee
Task Groups
Task Groups
Task Groups
Task Groups
Task Groups

Figure 1 Organization of CODASYL (January 1964 - July 1968)

The Language Subcommittee's function was much the same as was that of the former COBOL Maintenance Committee, namely, the maintenance and further development of COBOL. In addition it carried on liaison with the United States of America Standards Institute (USASI: formerly the American Standards Association - ASA) and the International Organization for Standardization (ISO) in their work concerning the development of proposed COBOL Standards.

The Publication Subcommittee was charged with the production of official COBOL publications and liaison with USASI as to the content of the COBOL Information Bulletin (CIB). The CIB is a collection of material relating to COBOL, distributed to the COBOL community by USASI.

The Evaluation Subcommittee's task was the analysis and evaluation of compiler implementations and user surveys. This Subcommittee provided information to the COBOL Committee regarding the use of COBOL.

The product of the COBOL Committee was the manual, "COBOL, Edition 1965".

Organizations participating in the work of the COBOL Committee were:

- Allstate Insurance Company
- American Telephone and Telegraph Company
- The Boeing Company
- Burroughs Corporation
- Canadian Federal Government
- Commercial Credit Corporation
- Collins Radio Company
- Control Data Corporation
- General Electric Company
- General Motors Corporation
- Honeywell, Incorporated
- International Business Machines Corporation
- Lockheed Aircraft Corporation
- National Bureau of Standards
- National Cash Register Company
- Owens-Illinois, Incorporated
- Philco Corporation
- Radio Corporation of America
- Southern Railway Company
- Standard Oil Company (N. J.)
- Sylvania Electric Products, Incorporated
- United States Air Force
- United States Bureau of the Budget
- United States Department of Army
- United States Navy
- United States Steel Corporation
- Univac Division of Sperry Rand Corporation
- Westinghouse Electric Corporation

2.1.4 PROGRAMMING LANGUAGE COMMITTEE

In July, 1968 the CODASYL Executive Committee adopted a revised constitution which accomplished certain needed organizational changes in an effort to stablize and improve the methods of achieving CODASYL objectives. This revised organization is shown in Figure 2.

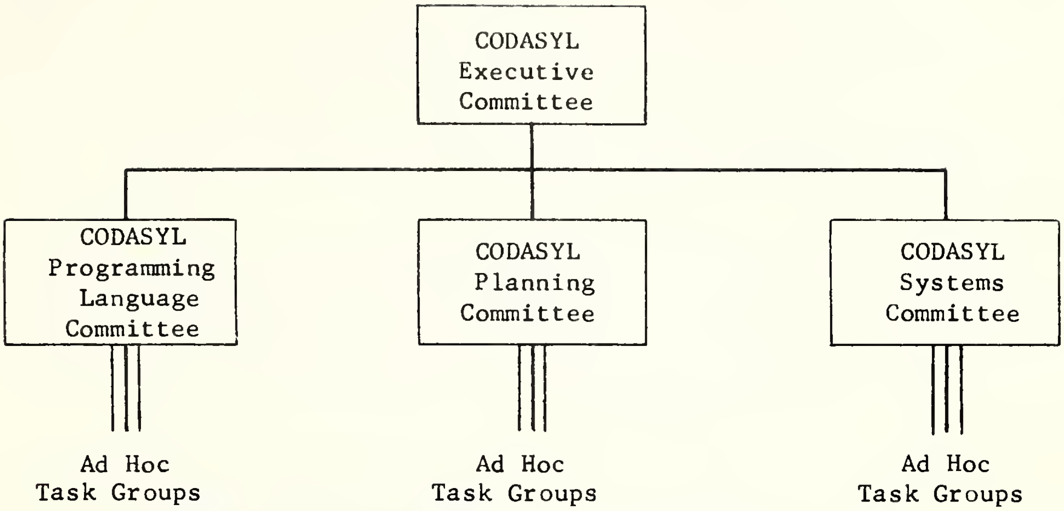

CODASYL Executive Committee
CODASYL Programming Language Committee
CODASYL Planning Committee
CODASYL Systems Committee
Ad Hoc Task Groups
Ad Hoc Task Groups
Ad Hoc Task Groups

Figure 2 Organization of CODASYL (Since July 1968)

With the formation of the Programming Language Committee (PLC) the former COBOL Language Subcommittee has been elevated to full committee status, and its chairman becomes a member of the Executive Committee.

The purpose and objectives of PLC include and extend those of the former CLS. The objectives are to make possible: compatible, uniform, source programs and object results, with continued reduction in the number of changes necessary for conversion or interchange of source programs and data. The PLC concentrates its efforts in the area of tools, techniques and ideas aimed at the programmer.

The Programming Language Committee is responsible for the presentation of the COBOL Journal of Development.

Member organizations of the COBOL Programming Language Committee are:

American Telephone and Telegraph Company
Burroughs Corporation
Canadian Federal Government
Control Data Corporation
General Electric Company
Honeywell, Incorporated
International Business Machines Corporation
National Bureau of Standards
National Cash Register Company
Radio Corporation of America

Southern Railway Company
United States Air Force
United States Bureau of the Budget
United States Department of Army
United States Navy
United States Steel Corporation
Univac Division of Sperry Rand Corporation
Westinghouse Electric Corporation

2.1.5 OTHER CODASYL ACTIVITIES

In 1961 a portion of the Intermediate Range Committee was combined with the Long Range Committee to form the Development Committee. This committee was comprised of a Systems Group and a Language Structures Group. In April 1965 the Development Committee was reorganized as the CODASYL Systems Committee (CSC) and CODASYL Language Structures Committee (CLSC) - as shown in Figure 1.

Before its reorganization, the Development Committee had produced:

a. In 1962, through the Systems Group, a "Decision Table Structured Language", identified as "Detab X".

b. In 1962, through the Language Structures Group, a "nonprocedural approach to problem statement", identified as "Information Algebra".

Under this revised committee structure, work continued on the further development of these and similar languages and techniques.

In July 1968 the CODASYL Executive Committee reorganized these committees resulting in the formation of the Planning Committee (replacing CLSC) and redefinition of the work of the Systems Committee. This structure is shown in Figure 2. These two committees have the following purposes:

a. The CODASYL Planning Committee is to aid in CODASYL planning by gathering, assimilating and disseminating information from implementors and users pertaining to the goals of CODASYL.

b. The CODASYL Systems Committee is to build an expertise in, and to develop advanced languages and techniques for, data processing, with the aim of automating as much as possible of the processes currently thought of as systems analysis, design and implementation.

2.2 EVOLUTION OF COBOL

2.2.1 COBOL-60

COBOL-60, the first version of the language published, proved that the concept of a common business oriented language was indeed practical.

2.2.2 COBOL-61

COBOL-61, the second official version of COBOL, was not completely compatible with COBOL-60. The changes were in areas such as organization of the Procedure Division rather than the addition of any major functions. The avowed goal of CODASYL in terms of successive versions of the language was to make changes of an evolutionary rather than revolutionary nature. This version was generally implemented and was the basis for many COBOL compilers.

The terms "Required" and "Elective" COBOL were adopted by the Executive Committee at this time in order to provide a uniform basis for a minimum implementation of COBOL. Those elements that were required to be implemented were designated as "Required" elements. Those elements of the language, the implementation of which it was permissible for the manufacturer to delay temporarily, were designated as "Elective" elements.

2.2.3 COBOL-61 EXTENDED

This version of COBOL was generally compatible with COBOL-61. The term "generally" must be used, not because of any basic changes in the philosophy or organization of the language, but because certain arithmetic extensions and general clarifications did make the syntax for certain statements and entries different from those in COBOL-61

COBOL-61 Extended, then, was generally COBOL-61 with the following major additions and modifications:

a. The addition of the Sort feature ,

b. The addition of the Report Writer option ,

c. The modification of the arithmetics to include multiple receiving fields and to add the CORRESPONDING option to the ADD and SUBTRACT statements , and

d. The inclusion of various clarifications.

2.2.4 COBOL, EDITION 1965

This version of COBOL included COBOL-61 Extended plus certain additions and modifications.

The major changes incorporated in COBOL, Edition 1965 were:

a. The inclusion of a series of options to provide for the reading, writing and processing of Mass Storage files,

b. The addition of the Table Handling feature which includes indexing and search options,

c. The modification of the specifications to delete the requirement for specific error diagnostic messages,

d. The deletion of the terms "Required" and "Elective", and

e. The inclusion of various clarifications.

2.2.5 COBOL, 1968

This version of COBOL, published in the Journal of Development, is based on COBOL, Edition 1965 with certain additions, and deletions.

The major changes incorporated in COBOL, 1968 are:

a. The inclusion of inter-program communication and the concept of a run unit,

b. The elimination of redundant editing clauses and certain data clauses more succinctly expressed by the PICTURE clause,

c. An improved COPY specification for all divisions except the identification division and the elimination of the INCLUDE statement,

d. The inclusion of a hardware independent means of specifying and testing for page overflow conditions,

e. The elimination of type 4 abbreviations,

f. The elimination of the DEFINE statement,

g. The inclusion of a remainder option for the DIVIDE statement,

h. The deletion of NOTE and REMARKS in favor of a general comment capability for all divisions,

i. The inclusion of the SUSPEND statement as additional means of controlling graphic display devices,

j. The inclusion of additional abbreviations,

k. A revision to the EXAMINE statement to allow the specification of dynamic parameter values, and

l. The inclusion of various clarifications.

2.3 STANDARDIZATION

The effort to define an International COBOL standard was begun in October of 1962 by the ISO Technical Committee 97, Computers and Information Processing, Working-Group E, Programming Languages. Working-Group E is now Subcommittee 5. The effort to define a United States COBOL standard was begun in September of 1962 by the ASA (now USASI) X3.4.4 Task Group on Processor Specifications and COBOL. This latter effort resulted in a USA Standard COBOL X3.23 which was approved in August of 1968. These standardization efforts are based on the technical content of COBOL as defined by CODASYL.

2.4 FUTURE DEVELOPMENTS

Further development and modification of COBOL are the responsibility of the Programming Language Committee of CODASYL. The development effort continues with emphasis on:

extended data base capabilities
input/output editing capabilities
a clarification and extension to asynchronous processing
a clarification and extension to the Report Writer facility
a communications facility
debugging facilities

SECTION II: PHILOSOPHY OF COBOL USE

TABLE OF CONTENTS

SECTION II: PHILOSOPHY OF COBOL USE

Page

CHAPTER 1. BACKGROUND II-1-1

CHAPTER 2. CONCEPTS II-2-1

2.1 Sort II-2-1
2.2 Report Writer II-2-2
2.3 Table Handling II-2-3
2.4 Mass Storage II-2-8
2.5 Rerun II-2-13
2.6 Same Area II-2-14
2.7 Inter-Program Communication II-2-14

CHAPTER 1

BACKGROUND

1.1 GENERAL

The task of specifying COBOL was undertaken because of the computer users' need for a problem-oriented, machine-independent language for business applications. Such a language, in order to be successful, should have certain characteristics. Programs written in the language should be capable of easy conversion from one machine to another, the language should be easily understandable in format and notation, it should provide good program documentation, and it should be capable of specifying data processing problems in such a way that an implementation of the language by means of a compiler can produce efficient object code.

COBOL is a language for programmers. It is not intended to be used by people unfamiliar with computers. The attributes of the language make it more feasible for use by applications oriented personnel and make it easier to teach to novices, but an understanding of programming and data processing concepts is prerequisite to the writing of efficient COBOL programs.

Although COBOL is a programmer's language, major benefits of the use of COBOL accrue to management by improving the over-all efficiency of the data processing function.

In describing a data processing problem, there are two elements involved. One is the set of procedures which specify how the data is to be manipulated, and the other is a description of the data involved. Furthermore, it was recognized that certain information pertaining to the specific computer on which the problem is to be run and some information identifying the program were also a necessary part of the description of a problem. The information pertaining to the computer itself would never carry over from one computer to another. However, it was felt that the advantages of having a common means of expression were sufficiently great to warrant the development of a standard form for even those items which clearly changed from computer to computer.

COBOL SYSTEM ELEMENTS

The COBOL system is composed of two elements, the source program written in COBOL and the compiler which translates this source program into an object program capable of running on a computer. This report, in general, considers only the source program and does not consider the compiler directly. However, the specifications of a language obviously determine, to a large extent, the boundaries of a compiler. Therefore, the compiler is mentioned in certain cases to facilitiate the explanation of the langauge.

A source program is used to specify the solution of a business data processing problem. The four elements of this specification are the:

1. Identification of the program.

2. Description of the equipment being used in the processing.

3. Description of the data being processed.

4. Set of procedures which determine how the data is to be processed.

The COBOL system has a separate division within the source program for each of these elements. The names of these divisions respectively are:

IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE

IDENTIFICATION DIVISION

The purpose of the Identification Division is to identify the source program and the outputs of a compilation. In addition, the user may include the date that the program was written, the date that the compilation was accomplished, and any other information which is desired.

ENVIRONMENT DIVISION

The Environment Division is that part of the source program which specifies the equipment being used. It contains descriptions of the computers to be used both for compiling the source program and for running the object program. Problem-oriented names may be assigned to particular equipment. Those aspects of a file which relate directly to hardware are described here. Because this division deals entirely with the specifications of the equipment being used, it is largely computer dependent.

DATA DIVISION

The Data Division uses file and record descriptions to describe the files of data that the object program is to manipulate or create and the individual logical records which comprise these files. The characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. Therefore, this division is to a large extent computer independent. While compatibility among computers cannot, in general, be absolutely assured, careful planning in the data layout will permit the same data description, with minor modifications, to apply to more than one computer.

PROCEDURE DIVISION

The Procedure Division specifies the steps that the user wishes the computer to follow. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This aspect of the over-all system is often referred to as the "program"; in reality it is only part of the total specification of the problem solution (i.e. the program) and is insufficient, by itself, to describe the entire problem. This is true because repeated references must be made, either explicitly or implicitly, to information appearing in the other divisions. This division, more than any other, allows the user to express his thoughts in meaningful English. Concepts of verbs to denote actions and sentences to describe procedures are basic, as is the use of conditional statements to provide alternative paths of action. The Procedure Division is essentially computer independent. That is, any user of COBOL can understand the information appearing in this division without regard to any particular computer. Furthermore, every COBOL compiler should interpret this information in the same way if the source program is syntactically correct.

1.1.1 DOCUMENTATION

The completeness of program documentation produced by a COBOL compilation is dependent upon the particular implementation (compiler) and upon the amount of thought and effort expended by the programmer. Because of the narrative nature and English-like syntax of the language, however, a certain amount of documentation is inherent. If the user establishes programmer guidelines so that data-names and procedure-names are consistent and meaningful, the listings produced by a compilation will constitute a good basis for a program documentation package. Such techniques as the use of the PICTURE clause for all data description at the elementary item level and the use of the COPY clause to call standard data descriptions or procedures from a library further enhance documentation. Generally, the quality and quantity of documentation produced by the use of COBOL is superior to that of other languages. This superiority is very important for the maintenance and revision of production programs.

1.1.2 UNDERSTANDABILITY

This feature of COBOL is in one respect closely allied with the previous topic of documentation. There is a good deal more to be said, however, for the relative ease of understanding a COBOL program.

The organization of the language, both as to data hierarchies and procedure statements, result in COBOL being much easier to teach and understand than machine-oriented languages or assembly systems. This attribute makes COBOL very useful for man-to-man as well as man-to-machine communications. However, it is necessary to keep in mind the importance of comprehensive training, including basic computer logic and programming fundamentals, as well as COBOL training itself.

In general, it is wise to keep the Procedure Division as simple as possible, avoiding the use of complex, nested, or compounded sentences, when possible.

1.1.3 COMPATIBILITY

While COBOL is not a completely machine-independent language, it closely approaches this goal. The user must exercise some degree of care if he wishes to produce highly compatible programs. It is appropriate here to examine, division by division, the degree of machine-independence in a COBOL program.

The Identification Division is totally machine-independent. However, since the ID Division does not produce object code, this machine-independence is not of great consequence to the user.

The Environment Division is almost completely machine-dependent. Those aspects of the data processing problem that are related to a particular hardware system are specified here. When a COBOL program is converted from one computer to another, this division normally would require at least a partial rewrite. The Environment Division usually consists of only three or four paragraphs, so that the effort of rewriting it is not large.

The Data Division is compatible across machine lines insofar as the data being described are compatible. For example, changes will be required in the Data Division if the user is converting from a BCD, variable word length machine to a fixed word length machine with more than one internal representation.

The Procedure Division is compatible across machine lines to the extent that the language can separate the problem from the hardware. If, for instance, the collating sequence of a new machine is different from that of the machine currently in use, certain statements in the Procedure Division may have to be rewritten. Aside from this type of "language independent" circumstance, the Procedure Division should convert from one machine to another with no change. The user can create incompatibilities if non-COBOL coding is introduced with the ENTER statement; if the COMPUTE statement is used to combine arithmetic operations, when the number of digits in the intermediate results of these arithmetic operations must be carefully controlled; or if implementor-dependent techniques are used.

This discussion of compatibility across machine lines assumes that features used in writing COBOL programs are present in both compilers.

1.1.4 EFFICIENCY

User experience has shown that there are many ways to achieve highly efficient COBOL object programs. In discussing the efficiency of COBOL, a distinction must be made between efficiency of compilation and efficiency of the object program.

As far as the speed of compilation is concerned, experience has shown that the most important factor here is the design of the compiler. Though the complexity and notational form of COBOL have some effect on compilation speed, examination of current implementations shows that compiling can be quite fast. Because the diagnostics included in the compiler and the documentation produced by the compiler significantly affect the measurement of compiling efficiency, they, as well as the factor of compilation speed, must be considered in this measurement.

In general, the efficiency of the object code produced is not limited by the structure, notation, or format of COBOL. Object program efficiency is mainly dependent on the compiler and the source programmer. Again, using current implementations as a guide, COBOL compilers are capable of producing efficient object programs, both in terms of running speed and in terms of memory usage.

One point of confusion that has arisen periodically during the history of COBOL use is language problems versus compiler problems. While it is true that language notation or structure can contribute to inefficient implementations, COBOL can, and has been, implemented efficiently.

With the dependence that exists between highly efficient object programs and programmer understanding of compiler idiosyncrasies, the implementor should supply his user with a manual containing specific guidelines outlining peculiarities of the compiler and methods for achieving most efficient programs. The user must exercise care in establishing the content of training courses and installation standards which result in compatible and efficient programs.

CHAPTER 2

CONCEPTS

COBOL offers many features which allow the user to obtain a necessary function without programming the function in detail. In this chapter we will discuss each of these features, considering the reason for its inclusion in the language and the concept of its use and organization.

2.1 SORT

Sorting has always constituted a large percentage of the workload in a business data processing shop. Therefore, an efficient sort program has always been a necessary part of any business software system.

In many sort applications it is necessary to apply some special processing to the contents of a sort file. The special processing may consist of addition, deletion, creation, altering, editing, or other modification of the individual records in the file. It may be necessary to apply the special processing before or after the records are reordered by the sort, or special processing may be required in both places. The COBOL Sort feature allows the user to express these procedures in the COBOL language and to specify at which point, before or after the sort, they are to be executed. A COBOL program may contain any number of sorts, and each of them may have its own independent special procedures. The Sort feature automatically causes execution of these procedures at the specified point in such a way that extra passes over the sort file are not required.

A Sort-file Description can be considered to be a particular type of File Description. That is, a sort-file, like any file, is a set of records.

The normal organization of a COBOL program containing a sort is such that the input file is read and operated upon by an INPUT PROCEDURE. Within this INPUT PROCEDURE, the RELEASE statement is used to create the sort-file. That is, at the conclusion of the INPUT PROCEDURE those records that have been output by use of the RELEASE statement rather than the WRITE statement comprise the sort-file, and this file is available only to the SORT statement. Execution of the SORT statement arranges the entire set of records in the sort file according to the keys specified in the SORT statement. The sorted records are made available from the sort-file by use of the RETURN statement during the OUTPUT PROCEDURE.

The sort-file has no label procedures which the programmer can control and the rules for blocking and for allocation of internal storage are peculiar to the SORT statement. The RELEASE and RETURN statements imply nothing with respect to buffer areas, blocks or reels. A sort-file, then, may be considered as an internal file which is created (RELEASE) from the input file, processed (SORT), and then made available (RETURN) to the output file. The sort-file itself is referred to and accessed only by the SORT statement.

2.2 REPORT WRITER

The production of reports of various types has always placed a large burden in terms of machine time and programmer time on the business data processing user. The need for a technique that would enable the programmer to specify and produce reports quickly and accurately in COBOL has long been felt. To this end the Report Writer feature has been added to the language.

The Report Writer allows the programmer to describe his report pictorially in the Data Division, thereby minimizing the amount of Procedure Division coding necessary.

In discussing the Report Writer, the physical aspects of the report format must be distinguished from the conceptual characteristics of the data in the report.

In describing the physical aspects of the report, consideration must be given to the hardware device on which the report is to be written and to the structure and format of the individual page. Facilities for specifying this information are included in the Report Writer entries.

The concept of a hierarchy of levels is used in defining the logical organization of the report. Each report is divided into report groups which in turn are divided into sequences of items. The use of a level structure permits the programmer to refer to an entire report name, major or minor report groups, elementary items within report groups, etc.

In creating the report the programmer defines the necessary report groups. A report group may be of any of the following types: heading group, footing group, control group, or detail print group. Further, a report group may extend over several actual lines on the page.

2.2.1 STRUCTURE

The report description entry contains information pertinent to the overall format of the named report and uses the level indicator RD. The characteristics of the report page are outlined by describing the number of physical lines per page and specifying the limits for presentation of headings, footings, and detail lines. Data items that act as format controls for a report are specified in the RD entry. Each report associated with an output file must be defined by an RD entry.

A report group is a set of data which may be made up of several print lines consisting of many data items or one print line containing only one data item. A report group description entry contains, in addition to other information, a level-number and a TYPE description. The level-number indicates the relative position in the data hierarchy of the report groups, and the TYPE clause describes the purpose of the report group in terms of its presentation within the report. If report groups are nested, that is, exist within groups, all groups within the assemblage must have the same TYPE description.

Specifically, the report group description entry defines the format and characteristics for a report group, whether this group is a series of lines, a line, or an elementary item. The relative placement of items within a report group, the level of a particular report group within the hierarchy of report groups, the format of all items, and any control factors associated with the group are defined in this entry.

Schematically, a report group is a line or a series of lines. The length of a line is determined by the compiler from environmental specifications. Initially, their lines consist of all SPACES. Within a report, the order of the individual report groups is not significant. Within a report group the programmer describes the elements consecutively from left to right and then from top to bottom. The description of a report group is analogous to the description of a data record, except that, in the report group spaces are assumed where no specific entry is indicated for presentation, while in the data record every character position must be explicitly defined, regardless of its data content.

2.3 TABLE HANDLING

Tables of data are common components of business data processing problems. Although the items that make up a table could be described as contiguous data items, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

Tables composed of contiguous data items are defined in COBOL by including the OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table-element and its name and description apply to each repetition or occurrence. Since each occurrence of a table-element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. The occurrence number is known as a subscript, and this technique of specifying individual table elements is called subscripting.

In order to facilitate such operations as table searching and manipulating specific items, a technique called Indexing is also available. Both subscripting and indexing are discussed below.

The number of occurrences of a table-element may be specified to be fixed or variable. If the occurrence number is given in the source program as fixed, the actual data that is entered into the table at object time may still comprise a variable number or occurrences of the table elements. Thus, not every table element need contain valid data.

2.3.1 TABLE DEFINITION

To define a one-dimensional table, the programmer uses an OCCURS clause as part of the data description of the table-element, but the OCCURS clause must not appear in the description of group items which contain the table-element. Example 1 shows a one-dimensional table defined by the item TABLE-ELEMENT.

Example 1.

```
01  TABLE-1.

    02  TABLE-ELEMENT; OCCURS 20 TIMES.

        03     DOG; ...

        03     FOX; ...
```

In Example 2, TABLE-ELEMENT defines a one-dimensional table, but DOG does not since there is an OCCURS clause in the description of the group item (TABLE-ELEMENT) which contains DOG.

Example 2.

```
02  TABLE-1.

    03  TABLE-ELEMENT; OCCURS 20 TIMES.

        04     DOG; OCCURS 5 TIMES

               05  EASY; ...

               05  FOX; ...
```

In both examples, the complete set of occurrences of TABLE-ELEMENT has been assigned the name TABLE-1. However, it is not necessary to give a group name to the table unless it is desired to refer to the complete table as a group item.

None of the three one-dimensional tables which appear in the following two examples have a group name.

Example 3.

```
     01  ABLE.

          02  BAKER; ...

          02  CHARLIE; OCCURS 20 TIMES

          02  DOG; ...
```

Example 4

```
     01  ABLE.

          02  BAKER; OCCURS 20 TIMES; ...

          02  CHARLIE; ...

          02  DOG; OCCURS 5 TIMES; ...
```

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that element. Thus, in Example 5, DOG is an element of a two-dimensional table; it occurs 5 times within each element of the item BAKER which itself occurs 20 times. BAKER is an element of a one-dimensional table.

Example 5.

```
     02  BAKER; OCCURS 20 TIMES; ...

          03  CHARLIE; ...

          03  DOG; OCCURS 5 TIMES; ...
```

In the general case, to define an n-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the descriptions of (n-1) group items which contain the element. In COBOL, tables of up to three dimensions are permitted; n cannot exceed 3 in the foregoing definition.

2.3.2 REFERENCES TO TABLE-ITEMS

Whenever the user refers to a table-element, or if the table-element is a group item, to the items within it, or to a conditon-name associated with the element or with items contained within the element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table the occurrence number of the desired element provides complete information. For tables of more than one dimension, an occurrence number must be supplied for each dimension of the table. In Example 5 then, a reference to the 4th BAKER or the 4th CHARLIE would be complete, whereas a reference to the 4th DOG would not. To refer to DOG, which is an element of a two-dimensional table, the user must refer to, for example, the 4th DOG in the 5th BAKER.

2.3.3 SUBSCRIPTING

One method by which occurrence numbers may be specified is to append one or more subscripts to the data-name. A subscript is an integer whose value specifies the occurrence number of an element within the group item that has the next lower level-number. The subscript can be represented either by a literal which is an integer or by a data-name which is defined elsewhere as a numeric elementary item with no character positions to the right of the assumed decimal point. In either case, the subscript, enclosed in parentheses, is written immediately following the name of the table element. A table element must include as many subscripts as there are dimensions in the table whose element is being referred to. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name, including the data-name itself.

Example 6.

```
02  BAKER; OCCURS 20 TIMES; ...

    03  CHARLIE; ...

    03  DOG; OCCURS 5 TIMES

        04  EASY; ...

        88  MAX; VALUE IS ...

        04  FOX; ...

            05  GEORGE; OCCURS 10 TIMES; ...

                06  HARRY; ...

                06  JIM; ...
```

In Example 6, references to BAKER and CHARLIE require only one subscript, references to DOG, EASY, MAX and FOX require two, and references to GEORGE, HARRY, and JIM require three.

When more than one subscript is required, they are written (separated by a space) in order corresponding to the occurrence numbers in successively less inclusive dimensions of the data organization. If a multi-dimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, then the subscripts are written from left to right in the order major, intermediate, and minor. Thus, in Example 6, a reference to HARRY (18, 2, 7) means the HARRY in the 7th GEORGE, in the 2nd DOG, in the 18th BAKER.

A reference to an item must not be subscripted if the item is not a table-element or an item or condition-name within a table-element.

The lowest permissible subscript value is 1. The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

When a data-name is used as a subscript, it may be used to refer to items within many different tables. These tables need not have elements of the same size. The data-name may also appear as the only subscript with one item and as one of two or three subscripts with another item. Also, it is permissible to mix literal and data-name subscripts, for example, HARRY (12, NEWKEY, 2).

2.3.4 INDEXING

Another method of referring to items in a table is indexing. To use this technique, the programmer assigns one or more index-names to an item whose data description contains an OCCURS clause. The INDEXED BY clause, by which the index-name is identified and associated with its table, is an optional part of the OCCURS clause. There is no separate entry to describe the index-name since its definition is completely hardware-oriented and it is not considered data per se. At object time the contents of the index-name will correspond to an occurrence number for that specific dimension of the table to which the index-name was assigned; however, the manner of correspondence will be determined by the implementor. The initial value of an index-name at object time is not determinable and the index-name must be initialized by the SET statement before use.

When a reference is made to a table-element, or to an item within a table-element, and the name of the item is followed by its related index-name or names in parentheses, then each occurrence number required to complete the reference will be obtained from the respective index-name. The index-name thus acts as a subscript whose value is used in any table reference that specifies indexing.

When a reference requires more than one occurrence number for completeness, the programmer must not use a data-name subscript to indicate one occurrence number and an index-name for another. Therefore, if indexing is to be used, each OCCURS clause within the hierarchy (each dimension of the table) must contain an INDEXED BY clause. The programmer may, however, mix literals and index-names within one reference, just as he may mix literals and data-name subscripts.

An index-name cannot be defined as part of a file, and therefore the index-name cannot be manipulated by input-output statements. Also, a data item in a file cannot be described as USAGE IS INDEX and no internal transfer, without conversion, between these data items and index-names can be accomplished by use of the SET statement.

At the time of execution of a statement which refers to an indexed table-element, the value of the index-name associated with the table-element must not correspond to a value less than 1 nor to a value greater than the highest permissible subscript value for the table-element.

The use of subscripting in a reference to a table-element, or to an item within a table-element, will not cause the alteration of any index-names associated with that table.

Relative indexing is an additional option for making references to a table-element or to an item within a table-element. When the name of the table-element is followed by an index in the form (index-name + integer-1), the occurrence number required to complete the reference will be the same as if integer-1 were added to the occurrence number to which the current setting at object time of the index-name corresponds. Similarly, when the form (index-name - integer-2) is used, the occurrence number obtained will be the same as if integer-2 were subtracted from the occurrence number to which the current setting of index-name corresponds.

It should be noted that the use of relative indexing will not cause the object program to alter the value of the index-name.

A reference to an item must not be indexed by an index-name that is not associated (through the use of the INDEXED BY clause) with the table of which that item is an element.

Data that has been arranged in the form of a table is very often searched. In COBOL the SEARCH statement provides facilities, through its two options, for producing serial and nonserial (for example, binary) searches. In using the SEARCH statement the programmer may vary an associated index-name or an associated data-name. This statement also provides facilities for execution of imperative statements when certain conditions are true and includes an AT END phrase.

2.4 MASS STORAGE

The operational characteristics and the processing requirements of mass storage devices differ significantly from those of magnetic tape, punched paper tape and punched cards. Tape and card files are normally organized in a sequential manner and the Data and Procedure Division of COBOL, prior to the inclusion of the mass storage facility, reflected this use.

Mass storage media can be used to store sequentially organized files, and this technique has been provided, but more significantly, mass storage devices have been designed to provide nonsequential storage and access capabilities.

The mass storage facility of COBOL includes specifications to provide for the effective use of mass storage devices. Mass storage clauses have been added to the Environment Division for describing the characteristics of mass storage files. The inclusion of a new Procedure Division statement SEEK, together with extensions to the specifications for the OPEN, READ, WRITE, and CLOSE statements provide the necessary facilities for the efficient processing of mass storage files.

2.4.1 ACCESS AND PROCESSING TECHNIQUES

The usual technique for applications using magnetic tape is sequential access to the data file and sequential processing of data records. This "sequential-sequential" technique is available for mass storage applications. Another technique for mass storage applications is called "random access and random processing." Either of these techniques, or a permutation of the two called "random-sequential", is specified by the programmer as the manner in which a particular mass storage file is to be processed.

A Mass Storage Control System, specified by each implementor, provides the mechanism for control of these techniques.

2.4.2 SEQUENTIAL ACCESS WITH SEQUENTIAL PROCESSING

Although this technique is similar in concept to the technique commonly used in processing magnetic tape files, there is a substantial difference between the physical environment of magnetic tape storage and the physical environment of mass storage.

In processing magnetic tape files the execution of a READ statement implies the possibility of physical movement of the tape reel and proper positioning of the reel for subsequent READ statement executions. This positioning is done without regard for execution of WRITE statements that reproduce the updated input record onto a physically different output file. In processing mass storage files, READ statements may refer to the same physical file as do the associated WRITE statements. That is, mass storage files are usually used for input and output at the same time. The usual file maintenance method is to read a record, process the record and return it to its previous location by means of a WRITE statement. Thus, once a record is located and read from a mass storage file, the record location may be retained and, when the record is returned to the file by the execution of a WRITE statement, the execution time for the WRITE statement may be greatly reduced.

An ACTUAL KEY clause, which specifies the actual hardware location of a specific mass storage record, is not required for the sequential-sequential technique. However, if the ACTUAL KEY clause is specified, varying the contents of the data item specified in the ACTUAL KEY clause (actual key) will not result in any variation in processing order. In the sequential-sequential mode the actual key is updated automatically by the implementor's Mass Storage Control System to reflect the location of the mass storage record currently being processed. Between the execution of the READ and WRITE statements for a particular file the contents of the actual key are static.

The execution of a READ statement followed logically by the execution of a WRITE statement for the same file results in an automatic updating of the actual key immediately after the execution of the WRITE statement. Similarly the execution of a WRITE statement followed logically by the execution of another WRITE statement, for the same file, results in an automatic updating of the actual key after the execution of each WRITE statement. However, the execution of a READ statement followed logically by the execution of another READ statement from the same file, without the intervening execution of a WRITE statement, results in the automatic updating of the actual key only immediately prior to the execution of the second READ statement. Following the execution of a WRITE statement the contents of the actual key reflects the actual location of the next mass storage record capable of being processed. In terms of COBOL logic, this is the location of the current mass storage record. Since the automatic updating of the contents of the actual key is the function of the implementor's Mass Storage Control System and since the ACTUAL KEY clause is never referred to or required by the Mass Storage Control System, any changes the programmer makes to the actual key do not affect the processing of the mass storage file.

The imperative statement in the AT END phrase associated with the next READ statement in order of execution is executed when the logical end of the mass storage file is detected. For WRITE statements the detection of the logical end of a mass storage file before the execution of the CLOSE statement causes the contents of the actual key to reflect a location outside the environmental limits of the file. As this value represents an erroneous location the file, the INVALID KEY phrase associated with a particular WRITE statement is executed when that WRITE statement is executed.

2.4.3 RANDOM ACCESS WITH SEQUENTIAL PROCESSING

In the sequential-sequential technique, the data records in a mass storage file are read, processed and written in an order based on the source program. The random-sequential technique differs only in that references are made to records in the file in a random manner. The sequential processing of randomly accessed records has all of the processing characteristics and file characteristics of the sequential-sequential method.

In order to permit direct access to any data record in a file, the programmer must specify to the Mass Storage Control System a key to the precise identification of the particular record desired. This identification must be in a format specified by the implementor for the particular hardware device being used. Since, in this technique, the control of an actual key is the responsibility of the programmer, there are no implicit updating functions for an actual key.

The introduction of the random access approach to a mass storage file requires the definition of an input-output statement (SEEK) to operate in conjunction with the READ and WRITE statements. In fact, the Mass Storage Control System must first locate the position of the desired record and then read or write the record. The function of locating the data record in the file is accomplished by the SEEK statement. Since, in the random-sequential technique the locating of records is always necessary, the function of the SEEK statement is performed implicitly by a READ or WRITE statement when the SEEK statement is not used by the programmer. The contents of the actual key are used by the Mass Storage Control System as the desired record's location identifier at the time of execution of the explicit or implicit SEEK statement.

The SEEK statement, then, locates a record for subsequent reading or writing. Other procedural statements may be executed during the physical seeking operation if they have been written between the SEEK statement and the READ statement for a particular file. The READ or WRITE of a particular record of a file can not be executed until the seeking operation has been completed.

There is a point concerning the SEEK and READ statements that must be recognized by the programmer. Until a READ statement is executed, any references to data items within the record description of the record being sought will refer to the contents of the last record obtained from the file. Therefore, if the programmer writes his Procedure Division to take advantage of the ability to execute statements during the seeking operation, he must take into account this "internal lag" of one record.

If the user has specified random access for a mass storage file, there is no logical end to the file. Thus, the AT END phrase of the READ statement is meaningless and the INVALID KEY phrase must be specified for both the READ and WRITE statements. If, during execution of either a READ or a WRITE statement, the contents of the actual key reflect an actual location outside the environmental limits for a file, the imperative statement in the INVALID KEY phrase is executed.

2.4.4 RANDOM ACCESS WITH RANDOM PROCESSING

If random processing is specified for a mass storage file, the Procedure Division references to the file occur in a USE FOR RANDOM PROCESSING Section in the Declaratives and the execution of this section is initiated by a PROCESS statement in the main program.

The use of the random-random technique requires the presence of the PROCESS statement with its associated Saved Area and out-of-line procedure. These functions, in turn, require an Asynchronous Control System (ACS) provided by the implementor. When a mass storage file is used for Random Processing and, consequently, is referred to in an out-of-line procedure, this control system takes on the coordinating functions required to control this technique.

Two levels of asynchronous processing capability may be posited. The first provides the capability to control the overlapping execution of several programs while the second, which may be thought of as a subset of the first, provides the capability to control the overlapping execution of several out-of-line procedures within the same program. The implementation of either level requires the specification of an ACS. The scope of the ACS as defined in this manual encompasses only the second level.

2.4.5 ASYNCHRONOUS PROCESSING

Asynchronous processing is the execution of a sequence of operations which are not necessarily completed in the order in which they were initiated. The implementation of this concept requires an ACS as part of the object program. Various implementor-specified asynchronous control systems may differ in the manner in which they control the execution of processing cycles, and in the manner and degree of coordination with input-output or real-time devices. The specifications for an ACS will be described with reference to the following language elements; the PROCESS and HOLD statements; the level indicator SA (Saved Area); and the Declarative Section USE FOR RANDOM PROCESSING.

The purpose of asynchronous processing is to achieve overlap of operations at object time and thus to reduce the time required to execute a sequence of operations asynchronously from that required to execute the same sequence of operations synchronously. To specify that a particular procedure is to be processed asynchronously the user must write that procedure within a USE FOR RANDOM PROCESSING Section. At the point in an in-line procedure at which the user wishes to initiate processing of the USE Section, he writes a PROCESS statement which is explicitly linked, in the source program, to that Section and to a Saved Area name.

Each Saved Area is identified in the File Section by the level indicator SA. The SA specification represents a memory area containing a number of records each of which may be any of the types described following the SA.

At any time during the execution of a procedure within the main line of the program rather than in a Declarative Section, only one of the record areas associated with a Saved Area is available for processing. This is the current record area and any reference from the in-line procedures to data within a particular Saved Area are treated as references to the current record area. Again, this is similar in concept to the manner in which references to data within a block of records for an input or output file are handled. In the case of asynchronous processing of mass storage files, however, the ACS is responsible for keeping track of the current record area.

2.4.6 CYCLE EXECUTION

The execution of a PROCESS statement initiates the execution of an out-of-line procedure. Each execution of a set of out-of-line procedures is termed a cycle. For each cycle a particular record area within the Saved Area becomes the current record area. When a particular cycle is completed, the current record area for that cycle is released for reassignment. Every cycle, then, is initiated by a PROCESS statement and ends when the end of the out-of-line procedure is reached. The cycle may be thought of as a closed subroutine that is executed once for each new record within the Saved Area.

There is one limitation on a cycle that does not apply to an ordinary closed subroutine. All data modified in the PROCESS loop and referred to by procedures within the cycle, must be in the current record area. Within this restriction the user may treat the cycle as if it were a closed subroutine performed in its entirety by the PROCESS statement.

The ACS controls the execution of a number of cycles within certain general limits. The ACS can suspend a cycle at any time so long as it keeps track of the point in the procedure at which the suspension took place and of the location of the current record area for that cycle. The ACS can reinstitute control in any previously suspended cycle by returning to the point of suspension in that cycle and re-establishing the location of the current record area for that cycle. The ACS can also return control to the in-line procedure providing at least one record area in the Saved Area is available.

The most general case is the one in which the cycles are completed in random order. Therefore, the user must write his program so that it operates correctly regardless of the order of completion of the cycle.

2.5 RERUN

The RERUN feature of COBOL provides a facility for check restart. That is, executing a RERUN takes a snapshot of the program status and stores the information. It is then possible to restart the program from the point of the most recent RERUN. The use of the RERUN clause protects the user from having to start a program over from the beginning in the event of a hardware failure while the job is running.

There are two basic parts to the RERUN clause. The user must designate a medium to receive the data and a criterion from which the frequency of checkpoints may be determined. The receiving medium may be specified by designating a file name or a separate hardware device. The determination of frequency of the dump may be made on the basis of a number of records of a particular file having been processed, of the end of a reel of a particular file having been reached, of the setting of a hardware switch or of a specified number of units of an internal clock having been counted.

2.6 SAME AREA

This feature is basically oriented toward saving memory space in the object program as it allows more than one file to share the same file area and alternate areas.

When used with the RECORD or SORT options of the SAME AREA clause, only the record area is shared and the alternate areas for each file remain independent. In this case any number of the files sharing the same record area may be active at one time. In implementations that include the RECORD option, this factor can give rise to an increase in the speed of the object program.

To illustrate this point, consider file maintenance. If the programmer assigns the same record area to both the old and new files, he not only saves memory in the object program, but because this technique eliminates a move of each record from the input to the output area, significant time savings result. An additional benefit of this technique is that the programmer need not define the record in detail as a part of both the old and new files. Rather, he defines the record completely in one case and simply includes the level 01 entry in the other. Because these record areas are in fact the same area, one set of names suffices for all processing requirements without requiring qualification.

When the SAME AREA clause is used without the RECORD or SORT option not only the file areas but the alternate areas as well, are shared.

As a result, only one of the files sharing the same set of areas is permitted to be active at one time. This form of the clause is designed for the application in which a series of files are used during different phases of the object program. In these cases, the SAME AREA clause allows the programmer to save memory space.

2.7 INTER-PROGRAM COMMUNICATION

It is frequently a convenience to be able to state a data processing problem as a set of inter-communicating programs. Such a facility allows independent compilation and debugging of the logical subdivisions of the problem, and it can reduce the coordination difficulties that arise when several programmers work on different parts of the problem.

In COBOL terminology, a program is either a source program or an object program depending on context; a source program is a syntactically correct set of COBOL statements as specified in Section III of this manual; an object program is the set of instructions, constants, and other machine-oriented data resulting from the operation of a compiler on a source program; and a run unit is the total machine language necessary to solve a data processing problem. It includes one or more object programs as defined above, and it may include machine language from sources other than a COBOL compiler.

When the statement of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other. This communication may take two forms: transfer of control and reference to common data.

2.7.1 TRANSFER OF CONTROL

The CALL statement provides the means whereby control can be passed from one program to another within a run unit. A program that is activated by a CALL statement may itself contain CALL statements. However, results are unpredictable where circularity of control is initiated; i.e., where program A calls program B, then program B calls program A or another program that calls program A.

When control is passed to a called program, execution proceeds in the normal way from procedure statement to procedure statement beginning with the first nondeclarative statement. If control reaches a STOP RUN statement, this signals the logical end of the run unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the point immediately following the CALL statement in the calling program. Stated briefly, the EXIT PROGRAM statement terminates only the program in which it occurs, and the STOP RUN statement terminates the entire run unit.

If the called program is not COBOL then the termination of the run unit or the return to the calling program must be programmed in accordance with the language of the called program.

2.7.2 INTER-PROGRAM DATA STORAGE

Program interaction requires that both programs have access to the same data items. In the calling program the common data items are described along with all other data items in the File Section, Working-Storage Section, Constant Section, or Linkage Section. At object time memory is allocated for the entire Data Division. In the called program, common data items are described in the Linkage Section. At object time memory space is not allocated for this section. Communication between the called program and the common data items stored in the calling program is effected through USING clauses contained in both programs. The USING clause in the calling program is contained in the CALL statement and the operands are a list of common data-identifiers described in its Data Division. The USING clause in the called program follows the Procedure Division header and the operands are a list of common data identifiers described in its Linkage Section. The identifiers specified by the USING clause of the CALL statement indicate those data items available to a calling program that may be referred to in the called program. The sequence of appearance of the identifiers in the USING clause of the CALL statement and the USING clause in the Procedure Division header is significant. Corresponding identifiers refer to a single set of data which is available to the calling program. The correspondance is positional, and not by name. While the called program is being executed, every reference to an operand whose identifier appears in the called program's USING clause is treated as if it were a reference to the corresponding operand in the USING clause of the active CALL statement.

Once control leaves a called program its state, if it is called again, is unpredictable. Therefore, initialization of the program in case of repetitive calls is the responsibility of the called program.

Execution of the CANCEL statement allows the user to indicate that the memory areas occupied by the called program(s) may be released.

SECTION III: COBOL LANGUAGE SPECIFICATIONS

TABLE OF CONTENTS

SECTION III. COBOL LANGUAGE SPECIFICATIONS

Page

CHAPTER 1. INTRODUCTION

III-1-1

1.1 Objectives of Section III III-1-1
1.2 History of COBOL Specification Documents III-1-1
1.3 Organization of Section III III-1-2
1.4 Notation Used in Formats and Rules III-1-2

CHAPTER 2. GLOSSARY

III-2-1

2.1 Introduction III-2-1
2.2 Definitions III-2-1

CHAPTER 3. LANGUAGE CONCEPTS

III-3-1

3.1 Character Set III-3-1
3.1.1 Categories III-3-1
3.1.2 Separators III-3-1
3.2 Character Strings III-3-2
3.2.1 Words III-3-2
3.2.1.1 Definition of Words III-3-2
3.2.1.2 Types of Words III-3-2
3.2.2 Literal III-3-6
3.2.3 PICTURE Character-String III-3-6
3.3 Concept of Computer-Independent Data Description III-3-7
3.3.1 Logical Record and File Concept III-3-7
3.3.2 Concept of Levels III-3-8
3.3.3 Concept of Classes of Data III-3-9
3.3.4 Selection of Character Representation and Radix III-3-10
3.3.5 Algebraic Signs III-3-10
3.3.6 Item Alignment for Increased Object-Code Efficiency III-3-10
3.3.7 Uniqueness of Data Reference III-3-11

CHAPTER 4. IDENTIFICATION DIVISION

III-4-1

4.1 General Description III-4-1
4.2 Organization III-4-1

TABLE OF CONTENTS

(CONTINUED)

Page

4.2.1 Structure III-4-1
4.3 The PROGRAM-ID Paragraph III-4-3
4.4 The DATE-COMPILED Paragraph III-4-4

CHAPTER 5. ENVIRONMENT DIVISION

III-5-1

5.1 General Description III-5-1
5.2 Organization III-5-1
5.2.1 Structure III-5-2
5.3 Configuration Section III-5-3
5.3.1 The SOURCE-COMPUTER Paragraph III-5-3
5.3.2 The OBJECT-COMPUTER Paragraph III-5-5
5.3.3 The SPECIAL-NAMES Paragraph III-5-7
5.4 Input-Output Section III-5-9
5.4.1 The FILE-CONTROL Paragraph III-5-9
5.4.2 The I-O-CONTROL Paragraph III-5-14

CHAPTER 6. THE DATA DIVISION

III-6-1

6.1 General Description III-6-1
6.2 File Section III-6-4
6.3 Record Description-Structure III-6-5
6.4 Working-Storage Section III-6-6
6.5 Constant Section III-6-8
6.6 Linkage Section III-6-10
6.7 Report Section III-6-12
6.8 The File Description--Complete Entry Skeleton III-6-14
6.9 The Sort File Description--Complete Entry Skeleton III-6-16
6.10 The Saved Area Description--Complete Entry Skeleton III-6-18
6.11 The Report Description--Complete Entry Skeleton III-6-20
6.12 The Data Description--Complete Entry Skeleton III-6-23
6.13 The Report Group Description--Complete Entry Skeleton . . . III-6-26
6.14 The AREA CONTAINS Clause III-6-29
6.15 The BLANK WHEN ZERO Clause III-6-30
6.16 The BLOCK Clause III-6-31
6.17 The CODE Clause III-6-33
6.18 The COLUMN NUMBER Clause III-6-34
6.19 The CONTROL Clause III-6-35
6.20 The Data-Name or FILLER Clause III-6-36
6.21 The DATA RECORDS Clause III-6-38
6.22 The GROUP INDICATE Clause III-6-39
6.23 The JUSTIFIED Clause III-6-40
6.24 The LABEL RECORDS Clause III-6-42
6.25 Level-Number III-6-43
6.26 The LINAGE Clause III-6-45
6.27 The LINE NUMBER Clause III-6-47
6.28 The NEXT GROUP Clause III-6-49

TABLE OF CONTENTS

(CONTINUED)

Page

6.29 The OCCURS Clause III-6-50
6.30 The PAGE LIMIT Clause III-6-53
6.31 The PICTURE Clause III-6-56
6.32 The RANGE Clause III-6-66
6.33 The RECORD CONTAINS Clause III-6-67
6.34 The RECORDING MODE Clause III-6-68
6.35 The REDEFINES Clause III-6-69
6.36 The RENAMES Clause III-6-71
6.37 The REPORT Clause III-6-73
6.38 The RESET Clause III-6-74
6.39 The SOURCE, SUM, and VALUE Clauses III-6-75
6.40 The SYNCHRONIZED Clause III-6-78
6.41 The TYPE Clause III-6-80
6.42 The USAGE Clause III-6-86
6.43 The VALUE Clause III-6-88
6.44 The VALUE OF Clause III-6-91

CHAPTER 7. THE PROCEDURE DIVISION III-7-1

7.1 General Description III-7-1
7.1.1 Declaratives III-7-1
7.1.2 Procedures III-7-1
7.1.3 Execution III-7-2
7.1.4 Procedure Division Structure III-7-2
7.2 Statements and Sentences III-7-4
7.2.1 Conditional Statements and Conditional Sentences III-7-4
7.2.2 Compiler Directing Statements and Compiler Directing Sentences III-7-4
7.2.3 Imperative Statements and Imperative Sentences III-7-5
7.3 Arithmetic Expressions III-7-6
7.3.1 Definition of an Arithmetic Expression III-7-6
7.3.2 Arithmetic Operators III-7-6
7.3.3 Formation and Evaluation Rules III-7-6
7.4 Conditions III-7-8
7.4.1 General Description III-7-8
7.4.2 Relation Condition III-7-8
7.4.3 Class Condition III-7-11
7.4.4 Condition-Name Condition III-7-12
7.4.5 Switch-Status Condition III-7-12
7.4.6 Sign Condition III-7-12
7.4.7 Compound Conditions III-7-13
7.4.8 Evaluation Rules III-7-16
7.5 Categories of Statements III-7-17
7.6 Common Options in Statement Formats III-7-19
7.6.1 The ROUNDED Option III-7-19
7.6.2 The SIZE ERROR Option III-7-19
7.6.3 The CORRESPONDING Option III-7-20
7.6.4 The Arithmetic Statements III-7-21

TABLE OF CONTENTS

(CONTINUED)

Page

7.6.5 Overlapping Operands III-7-21
7.6.6 Multiple Results in Arithmetic Statements III-7-21
7.7 The ACCEPT Statement III-7-22
7.8 The ADD Statement III-7-24
7.9 The ALTER Statement III-7-26
7.10 The CALL Statement III-7-27
7.11 The CANCEL Statement III-7-29
7.12 The CLOSE Statement III-7-30
7.13 The COMPUTE Statement III-7-35
7.14 The COPY Statement III-7-37
7.15 The DISPLAY Statement III-7-38
7.16 The DIVIDE Statement III-7-40
7.17 The ENTER Statement III-7-42
7.18 The EXAMINE Statement III-7-43
7.19 The EXIT Statement III-7-45
7.20 The GENERATE Statement III-7-46
7.21 The GO TO Statement III-7-48
7.22 The HOLD Statement III-7-50
7.23 The IF Statement III-7-51
7.24 The INITIATE Statement III-7-53
7.25 The MOVE Statement III-7-55
7.26 The MULTIPLY Statement III-7-58
7.27 The OPEN Statement III-7-60
7.28 The PERFORM Statement III-7-62
7.29 The PROCESS Statement III-7-70
7.30 The READ Statement III-7-72
7.31 The RELEASE Statement III-7-75
7.32 The RETURN Statement III-7-76
7.33 The SEARCH Statement III-7-78
7.34 The SEEK Statement III-7-82
7.35 The SET Statement III-7-83
7.36 The SORT Statement III-7-85
7.37 The STOP Statement III-7-90
7.38 The SUBTRACT Statement III-7-91
7.39 The SUSPEND Statement III-7-93
7.40 The TERMINATE Statement III-7-95
7.41 The USE Statement III-7-97
7.42 The WRITE Statement III-7-101

CHAPTER 8. SEGMENTATION III-8-1

8.1 General Description III-8-1
8.1.1 Scope III-8-1
8.1.2 Organization III-8-1
8.1.3 Segment Classification III-8-2
8.1.4 Segmentation Control III-8-2
8.2 Structure of Program Segments III-8-3
8.2.1 Priority-Numbers III-8-3

TABLE OF CONTENTS

(CONTINUED)

Page

8.2.2 Segment-Limit III-8-3

CHAPTER 9. THE COBOL LIBRARY III-9-1

9.1 Introduction III-9-1
9.2 The COPY Statement III-9-2

CHAPTER 10. REFERENCE FORMAT III-10-1

10.1 General Description III-10-1
10.2 Reference Format Representation III-10-1
10.3 Division, Section, Paragraph Formats III-10-3
10.4 Data Division Entries III-10-4
10.5 Declaratives III-10-4
10.6 Comment Lines III-10-5

CHAPTER 11. RESERVED WORDS III-11-1

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVES OF SECTION III

The specifications of a programming language are written for two principal groups: (1) the implementor of the language for a language processor (compiler) in a given hardware environment and, (2) the user of the language who will write source programs that can be translated for operation in a given hardware environment by the language processor. These specifications for a programming language are often two different sets but must reflect the same interpretive result, which is compatible understanding of the programming language definition.

Section III of this manual attempts to meet the needs of both the compiler-writing programmers and the source language programmers in that the rules of the language are described in a source program environment that is very nearly independent of hardware considerations. Where necessary, however, Section III describes the additional rules which must be followed by the user to interface with specific hardware units, or when the user should refer to other software manuals to determine dependent source program and other related entries.

1.2 HISTORY OF COBOL SPECIFICATION DOCUMENTS

Five official COBOL specification documents have been printed and released by the CODASYL Executive Committee. These are known as:

COBOL - 60, published 1960
COBOL - 61, published 1961
COBOL - 61 Extended, published 1963
COBOL, Edition 1965, published 1965
COBOL, Journal Of Development-1968, published 1968

The initial three documents principally addressed themselves to specifications for total definition of the language with interspersed paragraphs reflecting historical development, philosophy of the language, examples of the language use, etc. Section III of the later documents are a definition of the complete specifications with by-product paragraphs of philosophy, examples, etc. removed. The effect of Section III is to gather into one place all of the pertinent definitive rules separate from comments surrounding the specifications.

Whereas the COBOL - 60 specification is incompatible in language design with the COBOL - 61 specification, all of the subsequent specifications, including the 1968 COBOL edition, are based on the COBOL - 61 design. These include only extensions, resolutions of ambiguities, deletion of redundancies, or removal of unused or poor language specifications.

1.3 ORGANIZATION OF SECTION III

Section III is constructed to be a stand-alone document within the total presentation of COBOL. This recognizes that many readers of this document are mainly concerned with the COBOL language specifications and need not retain or maintain the other Sections.

Section III contains a specific chapter entitled 'Glossary'. For a complete understanding of terms used within Section III, specific definitions for COBOL usages are considered a basic part of the language definition. Experience has shown the developers, implementors, and users of the language that many incorrect interpretations, ambiguous understandings, and clarification problems resulted in the lack of a precise definition for the COBOL usage of a data processing term.

It is suggested that every reader become acquainted with the COBOL terminology expressed in the 'Glossary' to insure common understanding before attempting to interpret the COBOL specifications in later chapters.

1.4 NOTATION USED IN FORMATS AND RULES

1.4.1 DEFINITION OF A GENERAL FORMAT

A General Format is the specific arrangement of the elements of a clause or a statement. A clause or a statement consists of elements as defined below. Throughout this manual a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the General Formats. (Clauses that are optional must appear in the sequence shown if they are used.) In certain cases, stated explicitly in the rules associated with a given format, clauses may appear in sequences other than that shown. Applications, requirements or restrictions are shown as rules.

1.4.1.1 Syntax Rule

A syntax rule amplifies or restricts the usage of the elements within a general format.

1.4.1.2 General Rule

A general rule amplifies or restricts functions attributed to a general format or to its constituent elements.

1.4.1.3 Elements

Elements which make up a clause or a statement consist of upper case words, lower case words, level-numbers, brackets, braces, connectives, and special characters.

1.4.1.4 Words

All underlined upper-case words are called key words and are required when the functions of which they are a part are used. Upper-case words which are not underlined are optional to the user and may or may not be present in the source program. Upper-case words, whether underlined or not, must be spelled correctly.

Lower-case words, in a general format, are generic terms used to represent COBOL words that must be supplied by the user. Except for the list of words following, such lower-case words occurring in a general format are replaced, in an actual program, by COBOL words:

a. statement

b. imperative-statement

c. arithmetic-expression

d. character-string

e. comment-entry

f. condition

g. literal

These exceptions represent combinations of COBOL words constructed in accordance with the definitions specified in Chapter 7, Procedure Division for statement, imperative-statement, condition, and arithmetic-expression. Definition for the term character-string is given in Chapter 6, Data Division. Comment-entry is defined in Chapter 4, Identification Division. Literal is defined in Chapter 3, Language Concepts.

Where generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term for explanation or discussion

1.4.1.5 Level-Numbers

When specific level-numbers appear in data description entry formats, those specific level-numbers are required when such entries are used in a COBOL program.

1.4.1.6 Brackets and Braces

When a portion of a general format is enclosed in brackets, [ ], that portion may be included or omitted at the user's choice. Braces, { }, enclosing a portion of a general format means a selection of one of the options contained within the braces must be made. In both cases, a choice is indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of a format, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies (see 1.4.1.7, The Ellipsis).

1.4.1.7 The Ellipsis

In text, the ellipsis may show the omission of a portion of a source program. This meaning becomes apparent in context.

In the general format, the ellipsis represents the position at which repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows:

Given... in a clause or statement format, scanning right to left, determine the] or } delimiter immediately to the left of the ... ; continue scanning right to left and determine the logically matching [or { delimiter; the ... applies to the words between the determined pair of delimiters.

1.4.1.8 Format Punctuation

The punctuation characters, comma and semicolon, are shown in some formats. However, a semicolon must not appear immediately preceding the first clause of an entry or paragraph. The use of these punctuation characters for each division is as follows:

Identification Division
Although not expressly shown in the formats within this division, the comma and semicolon may be used within the comment-entries. The paragraph itself must terminate with a period followed by a space.

Environment Division
Where either a comma or a semicolon is shown in the formats, it is optional and may be included or omitted by the user. The entry itself must terminate with a period followed by a space.

Data Division
When either a comma or a semicolon is shown in the formats, it is optional and may be included or omitted by the user. The entry itself must terminate with a period followed by a space.

Procedure Division
When a comma is shown in the formats, the comma is optional and may be included or omitted by the user. If desired, a semicolon may be used between statements.

1.4.1.9 Use of Certain Special Characters in Formats

The characters '+', '-', '<', '>', '=', when appearing in formats, although not underlined, are required when such formats are used.

CHAPTER 2

GLOSSARY

2.1 INTRODUCTION

The terms in this chapter are defined in accordance with their meaning in COBOL, and may not have the same meaning for other languages.

These definitions are also intended to be either reference material or introductory material to be reviewed prior to reading the detailed language specifications that follow. For this reason, these definitions are, in most instances, brief and do not include detailed syntactical rules. Complete specifications for elements defined in this chapter can be located in other chapters of Section III.

2.2 DEFINITIONS

ACCESS, RANDOM

An access mode in which specific logical records are obtained from or placed in a mass storage file in a non-sequential manner under the control of an implementor's mass storage filing system.

ACCESS, SEQUENTIAL

An access mode in which a logical record read from or written to a file has an implicit logical predecessor and an implicit logical successor. The first access to a file accesses a record that has no implicit logical predecessor; each successive access refers to the implicit logical successor of the previously accessed logical record. The predecessor/successor relationships of a record are established when the record is written to a file.

ACTUAL DECIMAL POINT

(See DECIMAL POINT, ACTUAL)

ACTUAL KEY

(See KEY, ACTUAL)

ALPHABETIC CHARACTER

(See CHARACTER, ALPHABETIC)

ALPHANUMERIC CHARACTER

(See CHARACTER, ALPHANUMERIC)

AREA-NAME

A data-name that names a saved area.

AREA, SAVED

A storage area, specified in the Data Division, that is composed of one or more data records.

ARITHMETIC EXPRESSION

(See EXPRESSION, ARITHMETIC)

ARITHMETIC EXPRESSION CHARACTER

(See OPERATOR, ARITHMETIC)

ARITHMETIC OPERATOR

(See OPERATOR, ARITHMETIC)

ASCENDING KEY

(See KEY, ASCENDING)

ASSUMED DECIMAL POINT

(See DECIMAL POINT, ASSUMED)

ASYNCHRONOUS CONTROL SYSTEM

(See CONTROL SYSTEM, ASYNCHRONOUS)

ASYNCHRONOUS PROCESSING

(See PROCESSING, ASYNCHRONOUS)

BLOCK

A physical unit of data that is convenient to a particular computer for storage on an input or output device. The term is synonymous with Physical Record. The block is normally composed of one or more logical records, or a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block (see 3.3.1.2, Conceptual Characteristics of a File.)

CALLED PROGRAM

(See PROGRAM, CALLED)

CALLING PROGRAM

(See PROGRAM, CALLING)

CHARACTER

The basic indivisible unit of the language.

CHARACTER, ALPHABETIC

A character that belongs to the following set of letters:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space.

CHARACTER, ALPHANUMERIC

Any character in the computer's character set.

CHARACTER, ARITHMETIC EXPRESSION

(See OPERATOR, ARITHMETIC)

CHARACTER, EDITING

A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)

CHARACTER NUMERIC

A character that belongs to the following set of digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

CHARACTER PUNCTUATION

A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis
	space

CHARACTER, RELATION

A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

CHARACTER, SPECIAL

A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

CHARACTER SET

The complete COBOL character set consists of the 51 characters listed below:

Character	Meaning
0,1,..,9	digit
A,B,..,Z	letter
	space (blank)
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

CHARACTER-STRING

Contiguous characters which form a literal, a word or a PICTURE character-string. The rules governing the construction of each of the above types of character-strings differ, and are explained in other chapters of this section.

CHARACTERS, STANDARD

A character-string that comprises a data item whose size is measured in accordance with standard data format.

CLASS CONDITION

(See CONDITION, CLASS)

CLAUSE

A clause is an ordered set of consecutive COBOL words whose purpose is to specify an attribute of an entry.

CLAUSE, DATA

A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

CLAUSE, ENVIRONMENT

A clause that appears as part of an Environment Division entry.

CLAUSE, FILE

A clause that appears as part of any of the following Data Division entries:

File Description	(FD)
Sort File Description	(SD)
Report Description	(RD)
Saved Area Description	(SA)

COLLATING SEQUENCE

(See SEQUENCE, COLLATING)

COLUMN

A specific position within a report line.

COMMENT LINE

(See LINE, COMMENT)

COMPILE TIME

(See TIME, COMPILE)

COMPILER DIRECTING STATEMENT

(See STATEMENT, COMPILER DIRECTING)

CONDITION

A simple condition, or a syntactically correct combination of simple conditions and logical operators, for which a truth value can be determined.

CONDITION, CLASS

The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

CONDITION, CONDITION-NAME

The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

CONDITION, INVALID KEY

A condition in which, at object time, a specific value of the actual key associated with a mass storage file is determined to lie outside the limits of the file being accessed.

CONDITION, RELATION

The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item. (See Operator, Relational.)

CONDITION, SIGN

The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

CONDITION, SIMPLE

Any single condition chosen from the set:

relation condition
class condition
condition-name condition
switch-status condition
sign condition
(condition)

CONDITION, SWITCH-STATUS

The proposition, for which a truth value can be determined, that an implementor-defined switch, capable of being set to an ON or OFF status, has been set to a specific status.

CONDITIONAL STATEMENT

(See STATEMENT, CONDITIONAL)

CONDITIONAL VARIABLE

(See VARIABLE, CONDITIONAL)

CONDITION-NAME

The data-name assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the name assigned to a status of an implementor-defined device.

CONDITION-NAME CONDITION

(See CONDITION, CONDITION-NAME)

CONFIGURATION SECTION

(See SECTION, CONFIGURATION)

CONNECTIVE

A word or a punctuation character that is used to:

1. Associate a data-name or a paragraph-name with its qualifier.
2. Link two or more operands written in a series.
3. Form conditions (logical connectives). (See OPERATOR, LOGICAL).

CONSTANT

A unit of data whose value is not subject to change.

CONSTANT, FIGURATIVE

A reserved word that represents a numeric value, a character, or a string of characters.

CONSTANT, LITERAL

(See LITERAL)

CONSTANT, NAMED

A constant to which a data-name has been assigned.

CONSTANT SECTION

(See SECTION, CONSTANT)

CONTIGUOUS ITEMS

(See ITEMS, CONTIGUOUS)

CONTROL BREAK

The recognition of a change in the contents of a data item that has been designated as the data item that controls a hierarchy.

CONTROL DATA ITEM

(See DATA ITEM, CONTROL)

CONTROL FOOTING

(See FOOTING, CONTROL)

CONTROL GROUP

(See GROUP, CONTROL)

CONTROL HEADING

(See HEADING, CONTROL)

CONTROL HIERARCHY

A designated order of specific control data items.

CONTROL SYSTEM, ASYNCHRONOUS

An operating system that directs, or schedules, the execution of asynchronous processing cycles.

CONTROL SYSTEM, MASS STORAGE

An input-output control system that directs, or schedules, the processing of mass storage files.

COUNTER

A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

CURRENCY SIGN

The character '$' of the COBOL character set.

CURRENCY SYMBOL

The character defined by the CURRENCY-SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY-SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

DATA CLAUSE

(See CLAUSE, DATA)

DATA DESCRIPTION ENTRY

(See ENTRY, DATA DESCRIPTION)

DATA ITEM

Any elementary item, a named group of elementary items within a record, or a record.

DATA ITEM, CONTROL

A data item, described in the File Section or Working-Storage Section of a COBOL Source Program, that is associated with a control hierarchy and which, when a change in its contents is detected, causes a control break to be initiated.

DATA ITEM, INDEX

A data item in which the values associated with an index-name can be stored in a form specified by the implementor.

DATA-NAME

A word that contains at least one alphabetic character and that names an entry in the Data Division. When used in the General Formats, 'data-name' represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

DATA-NAME, INDEXED

An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

DATA-NAME, QUALIFIED

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

DATA-NAME, SUBSCRIPTED

An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

DECIMAL POINT, ACTUAL

The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

DECIMAL POINT, ASSUMED

A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

DECLARATIVES

A set of one or more special-purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the header DECLARATIVES and the last of which is followed by the header END DECLARATIVES. Each declarative operates under the control of either the in-line procedure or the implementor's input-output system or Report Writer and is composed of a section header, followed by a COPY or a USE compiler directing sentence, followed by a set of one or more associated paragraphs.

DESCENDING KEY

(See KEY, DESCENDING)

DIVISION

One or more sections or paragraphs that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four (4) divisions in a COBOL program:

IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE

DIVISION HEADER

(See HEADER, DIVISION)

EDITING CHARACTER

(See CHARACTER, EDITING)

ELEMENT, TABLE

A data item that belongs to the set of repeated items comprising a table.

ELEMENTARY ITEM

(See ITEM, ELEMENTARY)

END OF PROCEDURE DIVISION

The physical position in a COBOL source program after which no further procedures appear.

ENTRY

Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

ENTRY, DATA DESCRIPTION

An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

ENTRY, FILE DESCRIPTION

An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

ENTRY, OBJECT OF

A set of operands and reserved words, within a Data Division entry, that immediately follows the subject of the entry.

ENTRY, REPORT DESCRIPTION

An entry in the Report Section of the Data Division that is composed of the level indicator RD, followed by a data-name that is the name assigned to a particular report, and then followed by a set of file clauses as required.

ENTRY, SAVED AREA DESCRIPTION

An entry in the File Section of the Data Division that is composed of the level indicator SA, followed by a data-name that is the name assigned to a particular saved area, and then followed by a set of file clauses as required.

ENTRY, SORT FILE DESCRIPTION

An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

ENTRY, SUBJECT OF

An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

ENVIRONMENT CLAUSE

(See CLAUSE, ENVIRONMENT)

EXECUTION TIME

(See TIME, OBJECT)

EXPRESSION, ARITHMETIC

An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

FIGURATIVE CONSTANT

(See CONSTANT, FIGURATIVE)

FILE

A collection of records.

FILE CLAUSE

(See CLAUSE, FILE)

FILE DESCRIPTION ENTRY

(See ENTRY, FILE DESCRIPTION)

FILE LIMIT

A set of logical boundary locations for a particular mass storage file that are within the physical boundary locations of a mass storage medium.

FILE, MASS STORAGE

A collection of records that is assigned to a mass storage medium.

FILE, REPORT

A collection of records, produced by the Report Writer, whose content and format are such that they can be used for the preparation of a report.

FILE SECTION

(See SECTION, FILE)

FILE, SORT

A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

FILE-CONTROL

The name of an Environment Division paragraph in which the data files for a given source program are declared.

FILE-NAME

A data-name that names a file described in the Data Division.

FOOTING, CONTROL

A report group that occurs at the end of the control group of which it is a member and that is produced each time this control group is produced.

FOOTING, OVERFLOW

A report group that occurs at the end of a report page and that is produced before a page break, resulting from detection of a page limit condition (see 6.41, The TYPE Clause), is executed.

FOOTING, PAGE

A report group that occurs at the end of each report page and that is produced before a page break, resulting from detection of a page limit condition (see 6.41, The TYPE Clause), is executed.

FOOTING, REPORT

A report group that occurs at the end of a report and that is produced only once when the report is terminated.

FORMAT

A specific arrangement of a set of data.

FORMAT, REFERENCE

A format that provides a standard method for describing COBOL source programs.

FORMAT, REPORT

The format of a page in a particular report that is defined in the Report Section.

FORMAT, STANDARD DATA

The concept used in describing data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

GROUP, CONTROL

An integral set of related data that is specifically associated with a control data item in the report control hierarchy. The entire set of control heading report groups, control footing report groups, and associated detail groups comprise the control group for a given control data item.

GROUP ITEM

(See ITEM, GROUP)

GROUP, PRINT

(See GROUP, REPORT)

GROUP, REPORT

An integral set of related data within a report.

HEADER, DIVISION

A combination of reserved words followed by a period and a space that indicates the beginning of a division. The division headers are:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

HEADER, PARAGRAPH

A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers are:

In the Identification Division:

PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.

In the Environment Division:

SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.

HEADER, SECTION

A combination of words followed by a period and a space that indicates the beginning of a Section in the Environment, Data, and Procedure Division.

In the Environment and Data Division, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

In the Data Division:
FILE SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.
LINKAGE SECTION.
REPORT SECTION.

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a priority-number (optional), followed by a period and a space.

HEADING, CONTROL

A report group that occurs at the beginning of the control group of which it is a member and which is produced each time its control group is produced.

HEADING, OVERFLOW

A report group that occurs at the beginning of a report page and which is produced after a page break, resulting from detection of a page limit condition (see 6.41, The TYPE Clause) is executed.

HEADING, PAGE

A report group that occurs at the beginning of a report page and which is produced after a page break, resulting from detection of a page limit condition (see 6.41, The TYPE Clause) is executed.

HEADING, REPORT

A report group that occurs at the beginning of a report and that is produced only once when the report is initiated.

HIGH ORDER END

The leftmost character of a string of characters.

IDENTIFIER

A data-name, followed, as required, by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

IMPERATIVE STATEMENT

(See STATEMENT, IMPERATIVE)

IMPLEMENTOR-NAME

A word, specified by the implementor, that refers to a particular feature available on that implementor's computing system.

INDEX

A computer storage area or register, the contents of which represent the identification of a particular element in a table.

INDEX-NAME

A word with at least one alphabetic character that names an index associated with a specific table.

INDEX DATA ITEM

(See DATA ITEM, INDEX)

INDEXED DATA-NAME

(See DATA-NAME, INDEXED)

IN-LINE PROCEDURE

(See PROCEDURE, IN-LINE)

INPUT PROCEDURE

(See PROCEDURE, INPUT)

INPUT-OUTPUT SECTION

(See SECTION, INPUT-OUTPUT)

INTEGER

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in General Formats, integer must not be a numeric data item, and must be unsigned.

INVALID KEY CONDITION

(See CONDITION, INVALID KEY)

I-O-CONTROL

The name of an Environment Division paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device, are specified.

ITEMS, CONTIGUOUS

Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

ITEM, ELEMENTARY

A data item that is described as not being further logically subdivided.

ITEM, GROUP

A named contiguous set of elementary or group items.

ITEM, NONCONTIGUOUS

Data items, in the Working-Storage, Constant, or Linkage Section, that bear no hierarchic relationship to other noncontiguous items.

ITEM, NONNUMERIC

A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

ITEM, NUMERIC

A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9', with or without an operational sign.

KEY

One or more data items the contents of which jointly serve to identify the location of a record or the ordering of data.

KEY, ACTUAL

A key that directly expresses the physical location of a logical record on a mass storage medium.

KEY, ASCENDING

A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with a collating sequence.

KEY, DESCENDING

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with a collating sequence.

KEY WORD

(See WORD, KEY)

LEVEL INDICATOR

Two alphabetic characters that identify a specific type of file or a position in a hierarchy.

LEVEL-NUMBER

Two characters that in the case of the numbers 01 to 49, indicate the hierarchical structure of a logical record, or, in the case of the numbers 66, 77 and 88, identify special properties of a data description entry.

LIBRARY-NAME

A word that identifies a library entry which consists of a set of COBOL entries and/or procedures. The library-name must conform to the rules for formation of a procedure-name. The portion of the library-name actually used to interact with the COBOL library is specified by the implementor.

LINE, COMMENT

A source program line represented by an asterisk in the Continuation Area of the line and any characters from the computer's character set in Area A and Area B of that line. The Comment Line serves only for documentation in a program.

LINE, REPORT

A division of a page representing one row of characters.

LINKAGE SECTION

(See SECTION, LINKAGE)

LITERAL

A string of characters whose value is implied by the ordered set of characters comprising the string.

LITERAL, NONNUMERIC

A string of characters bounded by quotation marks. The string of characters may include any character in the computer's character set, with the exception of the quotation mark.

LITERAL, NUMERIC

A literal composed of one or more numeric characters, that also may contain either a decimal point, that cannot be the rightmost character, or an algebraic sign that must be the leftmost character, or both.

LITERAL CONSTANT

(See LITERAL)

LOGICAL OPERATOR

(See OPERATOR, LOGICAL)

LOGICAL RECORD

(See RECORD, LOGICAL)

LOW ORDER END

The right most character of a string of characters.

MASS STORAGE

A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

MASS STORAGE CONTROL SYSTEM

(See CONTROL SYSTEM, MASS STORAGE)

MASS STORAGE FILE

(See FILE, MASS STORAGE)

MASS STORAGE FILE SEGMENT

A part of a mass storage file whose beginning and end is defined by the FILE-LIMITS clause in the Environment Division.

MNEMONIC-NAME

A word, supplied by the programmer, that is associated in the Environment Division with a specific implementor name.

MODE-NAME

A word, specified by the implementor, that refers to a particular method of data representation on a physical storage medium.

NAMED CONSTANT

(See CONSTANT, NAMED)

NONCONTIGUOUS ITEM

(See ITEM, NONCONTIGUOUS)

NONNUMERIC ITEM

(See ITEM, NONNUMERIC)

NONNUMERIC LITERAL

(See LITERAL, NONNUMERIC)

NUMERIC CHARACTER

(See CHARACTER, NUMERIC)

NUMERIC ITEM

(See ITEM, NUMERIC)

NUMERIC LITERAL

(See LITERAL, NUMERIC)

OBJECT-COMPUTER

The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

OBJECT OF ENTRY

(See ENTRY, OBJECT OF)

OBJECT PROGRAM

(See PROGRAM, OBJECT)

OBJECT TIME

(See TIME, OBJECT)

OPERAND

Any lower case word (or words) that appears in a statement or entry format in this publication.

OPERATIONAL SIGN

(See SIGN, OPERATIONAL)

OPERATOR, ARITHMETIC

A single character, or a fixed two-character combination for the character(s) that belongs to the following set:

Character	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

OPERATOR, LOGICAL

One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND or OR, or both, can be used as logical connectives. NOT can be used for logical negation.

OPERATOR, RELATIONAL

A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meaning are:

RELATIONAL OPERATOR	MEANING
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to
IS UNEQUAL TO	Not equal to
EQUALS	Equal to
EXCEEDS	Greater than

OPERATOR, UNARY

A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1 respectively.

OPTIONAL WORD

(See WORD, OPTIONAL)

OUT-OF-LINE PROCEDURE

(See PROCEDURE, OUT-OF-LINE)

OUTPUT PROCEDURE

(See PROCEDURE, OUTPUT)

OVERFLOW FOOTING

(See FOOTING, OVERFLOW)

OVERFLOW HEADING

(See HEADING, OVERFLOW)

PAGE

A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

PAGE FOOTING

(See FOOTING, PAGE)

PAGE HEADING

(See HEADING, PAGE)

PARAGRAPH

A paragraph-name (in the Procedure Division) followed by one or more sentences, or a paragraph-header (in the Identification Division and the Environment Division) followed by one or more entries.

PARAGRAPH HEADER
(See HEADER, PARAGRAPH)

PARAGRAPH-NAME

A word that identifies and begins a paragraph in the Procedure Division.

PHYSICAL RECORD

(See BLOCK)

PRINT GROUP

(See GROUP, REPORT)

PRIORITY-NUMBER

A number, ranging in value from '0' to '99', that classifies source program sections in the Procedure Division in order to guide object program segmentation.

PROCEDURE

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

PROCEDURE-NAME

A word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name (which may be qualified), or a section-name.

PROCEDURE, IN-LINE

The set of statements that constitutes the main or controlling flow of the program, and which exclude statements under control of the asynchronous control system.

PROCEDURE, INPUT

A set of statements that is executed each time a record is released to the sort file.

PROCEDURE, OUT-OF-LINE

A set of statements not included in the main or controlling flow of the run-unit.

PROCEDURE, OUTPUT

A set of statements that is executed each time a sorted record is returned from the sort file.

PROCESSING, ASYNCHRONOUS

The manner of processing logical records within out-of-line procedures that are initiated in consecutive relation to one another but that are not necessarily executed or completed in the order in which they are initiated; in other words, no specific processing cycle is necessarily completed before a subsequent cycle is initiated.

PROCESSING CYCLE

A single execution of a defined out-of-line procedure.

PROCESSING, RANDOM

(See PROCESSING, ASYNCHRONOUS)

PROCESSING, SEQUENTIAL

(See PROCESSING, SYNCHRONOUS)

PROCESSING, SYNCHRONOUS

The manner of processing logical records within in-line procedures in the order in which the records are made available.

PROGRAM, CALLED

A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

PROGRAM, CALLING

A program which executes a CALL to another program.

PROGRAM, OBJECT

A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program'.

PROGRAM, SOURCE

Although it is recognized that a source program may be represented by other forms and symbols, in this report it always refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'source program'.

PROGRAM-NAME

A word that identifies a COBOL source program.

PUNCTUATION CHARACTER

(See CHARACTER, PUNCTUATION)

QUALIFIED DATA-NAME

(See DATA-NAME, QUALIFIED)

QUALIFIER

A data-name that names, or a section-name that makes a non-unique data-name at a lower level in the same hierarchy or a non-unique paragraph-name, respectively, unique.

RANDOM ACCESS

(See ACCESS, RANDOM)

RANDOM PROCESSING

(See PROCESSING, ASYNCHRONOUS)

RECORD

(See RECORD, LOGICAL)

RECORD DESCRIPTION

The total set of data description entries associated with a particular record.

RECORD, LOGICAL

The most inclusive data item.

RECORD, PHYSICAL

(See BLOCK)

RECORD-NAME

A data-name that names a record.

REFERENCE FORMAT

(See FORMAT, REFERENCE)

REGISTERS, SPECIAL

Compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

RELATION

(See OPERATOR, RELATIONAL)

RELATION CHARACTER

(See CHARACTER, RELATION)

RELATION CONDITION

(See CONDITION, RELATION)

RELATIONAL OPERATOR

(See OPERATOR, RELATIONAL)

REPORT

A presentation of a set of data described in a Report File.

REPORT-NAME

A data-name that names a report.

REPORT DESCRIPTION ENTRY

(See ENTRY, REPORT DESCRIPTION)

REPORT FILE

(See FILE, REPORT)

REPORT FOOTING

(See FOOTING, REPORT)

REPORT FORMAT

(See FORMAT, REPORT)

REPORT GROUP

(See GROUP, REPORT)

REPORT HEADING

(See HEADING, REPORT)

REPORT LINE

(See LINE, REPORT)

REPORT SECTION

(See SECTION, REPORT)

RESERVED WORD

(See WORD, RESERVED)

RUN UNIT

(See UNIT, RUN)

SAVED AREA

(See AREA, SAVED)

SAVED AREA DESCRIPTION ENTRY

(See ENTRY, SAVED AREA DESCRIPTION)

SECTION

A set of one or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

SECTION, CONFIGURATION

A section of the Environment Division that describes overall specifications of source and object computers.

SECTION, CONSTANT

The section of the Data Division that deals with named constants, written as noncontiguous items, or grouped into constant records.

SECTION, FILE

The section of the Data Division that contains file description entries.

SECTION HEADER

(See HEADER, SECTION)

SECTION, INPUT-OUTPUT

The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

SECTION, LINKAGE

The section in the Data Divisions of the called programs that describes data items available from the calling program. These data items may be referred to by both the calling and the called program.

SECTION, REPORT

The section of the Data Division that contains one or more report description entries.

SECTION, WORKING-STORAGE

The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

SECTION-NAME

A word that identifies a section written in the Procedure Division. (See WORD)

SENTENCE

A sequence of one or more statements, the last of which is terminated by a period followed by a space.

SEPARATOR

The space and optional punctuation characters used as delimiters to enhance readability and to eliminate ambiguity.

SEQUENCE, COLLATING

The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting and comparing.

SEQUENTIAL ACCESS

(See ACCESS, SEQUENTIAL)

SEQUENTIAL PROCESSSING

(See PROCESSING, SYNCHRONOUS)

SIGN CONDITION

(See CONDITION, SIGN)

SIGN, OPERATIONAL

An algebraic sign, associated with a numeric data item, to indicate whether the item is positive or negative.

SIMPLE CONDITION

(See CONDITION, SIMPLE)

SORT FILE

(See FILE, SORT)

SORT FILE DESCRIPTION ENTRY

(See ENTRY, SORT FILE DESCRIPTION)

SOURCE PROGRAM

(See PROGRAM, SOURCE)

SOURCE-COMPUTER

The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

SPECIAL CHARACTER

(See CHARACTER, SPECIAL)

SPECIAL REGISTERS

(See REGISTERS, SPECIAL)

SPECIAL-NAMES

The name of an Environment Division paragraph in which implementor-names are related to user-specified mnemonic-names.

STANDARD CHARACTERS

(See CHARACTERS, STANDARD)

STANDARD DATA FORMAT

(See FORMAT, STANDARD DATA)

STATEMENT

A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

STATEMENT, COMPILER DIRECTING

A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

STATEMENT, CONDITIONAL

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

STATEMENT, IMPERATIVE

A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements (see 7.2.3, Imperative Statements and Imperative Sentences).

SUBJECT OF ENTRY

(See ENTRY, SUBJECT OF)

SUBPROGRAM

(See PROGRAM, CALLED)

SUBSCRIPT

An integer whose value identifies a particular element in a table.

SUBSCRIPTED DATA-NAME

(See DATA-NAME, SUBSCRIPTED)

SWITCH-STATUS CONDITION

(See CONDITION, SWITCH-STATUS)

SYNCHRONOUS PROCESSING

(See PROCESSING, SYNCHRONOUS)

TABLE

A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

TABLE ELEMENT

(See ELEMENT, TABLE)

TIME, COMPILE

The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

TIME, EXECUTION

(See TIME, OBJECT)

TIME, OBJECT

The time at which an object program is executed.

TRUTH VALUE

The representation of the result of the evaluation of a condition in terms of one of two values:

true
false

UNARY OPERATOR

(See OPERATOR, UNARY)

UNIT

A module of mass storage the dimensions of which are determined by each implementor.

UNIT, RUN

A set of one or more object programs which function, at object time, as a unit to provide problem solutions.

VARIABLE

A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

VARIABLE, CONDITIONAL

A data item one or more values of which has a condition-name assigned to it.

VERB

A word that expresses an action to be taken by a COBOL compiler or object program.

WORD

A sequence of not more than 30 characters. Each character is selected from the set 'A', 'B', 'C',...'Z', '0'...'9', '-' except that the '-' may not appear as the first or last character in a word. A word is delimited by separators.

WORD, KEY

A reserved word whose presence is required when the format in which the word appears is used in a source program.

WORD, OPTIONAL

A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

WORD, RESERVED

One of a specified list of words which may be used in a COBOL source program, but which must not appear in the programs as user-defined words.

WORKING-STORAGE SECTION

(See SECTION, WORKING-STORAGE)

CHAPTER 3

LANGUAGE CONCEPTS

3.1 CHARACTER SET

The complete character set for COBOL consists of the 51 characters defined under CHARACTER SET in Chapter 2, Glossary. In the discussions that follow, all references are to Chapter 2, Glossary.

3.1.1 CATEGORIES

The character set for words consists of the characters defined under WORD.

The character set for punctuation consists of the characters defined under CHARACTER, PUNCTUATION.

The character set for arithmetic operators consists of the characters defined under OPERATOR, ARITHMETIC.

The special characters for relational operators are defined under CHARACTER, RELATION.

The character set for editing consists of the characters defined under CHARACTER, EDITING.

Those characters that are recognized within COBOL include the letters of the alphabet, digits, and those characters, commonly called symbols, which are used in expressions, relations and editing. Since the character set of a particular computer may not have the characters defined above, single character substitution must be made as required. When such a character set contains fewer than 51 characters, double characters must be substituted for the single characters.

3.1.2 SEPARATORS

The space and the punctuation characters, when not contained within quotation marks are separators. Where a space is used, more than one may be used, except for the restrictions set forth in this chapter and in Chapter 10, Reference Format.

A character-string is delimited on the right by a space, period, right parenthesis, comma, or semicolon. The use of punctuation characters in connection with character-string is defined as follows:

1. A space must follow a period, comma and semicolon when any of these punctuation characters are used to delimit character-string,
2. A space may immediately follow a left parenthesis or may immediately precede a right parenthesis, and
3. When used as punctuation characters, period, comma or semicolon may be preceded by a space except as required by special insertion editing in the PICTURE clause (see 6.31.5, Editing Rules).

3.2 CHARACTER STRINGS

A character-string is a sequence of contiguous characters which form a literal, a word or a PICTURE character-string.

3.2.1 WORDS

3.2.1.1 Definition of Words

A word is a sequence of not more than 30 characters. Each character is selected from the set 'A', 'B', 'C', ... 'Z', '0' ... '9', '-' except that the '-' may not appear as the first or last character in a word.

3.2.1.2 Types of Words

3.2.1.2.1 Data-Name

A data-name is a word that contains at least one alphabetic character and that names an entry in the Data Division.

3.2.1.2.2 Condition-Name

A condition-name is a word with at least one alphabetic character, which is assigned to a specific value, set of values or range of values, within the complete set of values that a data item may assume. The data item itself is called a conditional variable. Each condition-name must be unique, or be made unique through qualification. A conditional variable may be used as a qualifier for any of its condition-names. If references to a conditional variable require indexing, subscripting or qualification, then references to any of its condition-names also require the same combination of indexing, subscripting or qualification (see 3.3.7, Uniqueness of Data Reference).

In addition to being described in the Data Division, condition-names may also be defined in the SPECIAL-NAMES paragraph within the Environment Division, where a condition-name must be given to the ON status or OFF status, or both, of implementor-defined switches.

A condition-name is used in conditions as an abbreviation for the relation condition; this relation condition posits that the associated conditional variable is equal to one of the set of values to which that condition-name is assigned.

3.2.1.2.3 Procedure-Name

A procedure-name is a word which is used to name a paragraph or section in the Procedure Division. Procedure-names composed only of the digits '0' through '9' are equivalent if, and only if, they are composed of the same number of digits and have the same value.

3.2.1.2.4 Figurative Constants

Certain constants, called figurative constants, have been assigned fixed data-names. These data-names must not be bounded by quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The fixed data-names and their meanings are as follows:

ZERO ZEROS ZEROES	Represents the value '0', or one or more of the character '0'. depending on context.
SPACE SPACES	Represents one or more blanks or spaces.
UPPER-BOUND UPPER-BOUNDS	Represents one or more of the characters conventionally used as a high delimiter in processing data. It can be, but it is not necessarily, the character with the highest value in each computer's collating sequence.
LOWER-BOUND LOWER-BOUNDS	Represents one or more of the characters conventionally used as a low delimiter in processing data. It can be, but it is not necessarily, the character with the lowest value in each computer's collating sequence.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest value in each computer's collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest value in each computer's collating sequence.
QUOTE QUOTES	Represents one or more of the character '"' or the character that has been substituted for it on computers whose character set does not contain a quotation mark. The word QUOTE cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTE ABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD".
ALL literal	Represents one or more of the string of characters comprising the literal. The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

1. When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is equal to the size in characters of the associated data item.
2. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY, EXAMINE or STOP statement, the length of the string is one character. The figurative constant ALL literal may not be used with DISPLAY, EXAMINE or STOP.

A figurative constant can be used any place where a literal appears in the format, except that whenever the literal is restricted to having only numeric characters in it, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

3.2.1.2.5 Special Registers

1. TALLY

The word TALLY is the name of a special register whose implicit description is that of an integer of five digits without an operational sign, and whose implicit USAGE is COMPUTATIONAL. The primary use of the TALLY register is to hold information produced by the EXAMINE statement. The word TALLY may also be used as a data-name wherever an elementary data item of integral value may appear.

2. LINE-COUNTER

The word LINE-COUNTER is the fixed data-name for a line-counter that is used by the Report Section in the Data Division to generate automatically PAGE/OVERFLOW HEADING and PAGE/OVERFLOW FOOTING report groups. One line-counter is automatically supplied for each report described in the Report Section if a PAGE LIMIT clause is written in the Report Description entry. (See 6.11.6, LINE-COUNTER rules.)

3. PAGE-COUNTER

The word PAGE-COUNTER is a fixed data-name for a page-counter generated by the Report Section for use as a source data item to present page numbers within a report group. One page-counter is supplied for each report by the Report Section if the word PAGE-COUNTER is given as a source data item in a Report Group Description entry. (See 6.11.5, PAGE-COUNTER rules.)

4. LINAGE-COUNTER

The word LINAGE-COUNTER is a fixed identifier for a line-counter generated by the presence of a LINAGE clause in a File Description. The implicit description is that of an integer with size based on the number of lines specified per page in the LINAGE clause. The value represented in the LINAGE-COUNTER at any given time is the number of lines advanced within a printed page. One LINAGE-COUNTER is supplied for each file in the File Section if the LINAGE clause appears in the FD entry (see 6.26, The LINAGE Clause).

3.2.1.2.6 Mnemonic Names

Mnemonic names are the means of relating implementor-names with problem-oriented names, and, also, the status of switches with condition-names. (See 5.3.3, The SPECIAL-NAMES Paragraph.)

3.2.1.2.7 Reserved Words

A specified list of words which may be used in a COBOL source program, but which must not appear in the programs as user-defined words (see Chapter 11, Reserved Words).

There are three types of reserved words, as shown below.

1. Key Words

   A key word is a word whose presence is required when the format in which the word appears is used in a source program. Within each format, such words are uppercase and underlined.

   Key words are of three types:

   a. Verbs such as ADD, READ, and ENTER.

   b. Required words, which appear in statement and entry formats.

   c. Words which have a specific functional meaning such as NEGATIVE, SECTION, TALLY, etc.

2. Optional Words

   Within each format, uppercase words that are not underlined are called optional words and may appear at the user's option. The presence or absence of each optional word within a format does not alter the compiler's translation. Misspelling of an optional word, or its replacement by another word of any kind is not allowed.

3. Connectives

   There are three types of connectives:

   a. Qualifier connectives that are used to associate a data-name or a paragraph-name with its qualifier:
      OF, IN.

   b. A series connective that links two or more consecutive operands:
      , (comma).

   c. Logical connectives that are used in the formation of conditions:
      AND, OR, AND NOT, OR NOT.

CHARACTER-STRING

3.2.2 LITERAL

A literal is a string of characters whose value is implied by an ordered set of characters of which the literal is composed. Every literal belongs to one of two types, numeric or nonnumeric.

A nonnumeric literal is defined as a string of any allowable characters in the computer's character set, excluding the character quotation mark, of anv length, bounded by quotation marks. The value of a nonnumeric literal is the string of characters itself, excluding the quotation marks. Any spaces enclosed in the quotation marks are part of the nonnumeric literal and, therefore, are part of the value. All nonnumeric literals are category alphanumeric (see 6.31, The PICTURE Clause).

A numeric literal is defined as a string of characters chosen from the digits '0' thru '9', the plus sign, the minus sign, and the decimal point; the rules for the formation of numeric literals are as follows:

1. A literal must contain at least one digit.

2. A literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.

3. A literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

   The word 'integer' appearing in a general format represents a numeric literal containing no decimal point.

   If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal and it is treated as such by the compiler.

4. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric (see 6.31, The PICTURE Clause).

3.2.3 PICTURE CHARACTER-STRING

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. The allowable combinations are explained under the PICTURE clause (see 6.31, The PICTURE Clause).

3.3 CONCEPT OF COMPUTER-INDEPENDENT DATA DESCRIPTION

To make data as computer-independent as possible, the characteristics or properties of the data are described in relation to a Standard Data Format rather than an equipment-oriented format. This Standard Data Format is oriented to general data processing applications and uses the decimal system to represent numbers (regardless of the radix used by the computer) and the remaining characters in the COBOL character set to describe nonnumeric data items.

3.3.1 LOGICAL RECORD AND FILE CONCEPT

The approach taken in defining file information is to distinguish between the physical aspects of the file and the conceptual characteristics of the data contained within the file.

3.3.1.1 Physical Aspects of a File

The physical aspects of a file describe the data as it appears on the input or output media and include such features as:

1. The mode in which the data file is recorded on the external medium.

2. The grouping of logical records within the physical limitations of the file medium.

3. The means by which the file can be identified.

3.3.1.2 Conceptual Characteristics of a File

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit.

A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and bears no direct relationship to the size of the file of information contained on a device.

A logical record may be contained within a single physical unit; or several logical records may be contained within a single physical unit; or a logical record may require more than one physical unit to contain it. There are several source language methods available for describing the relationship of logical records and physical units. Once the relationship has been established, the control of the accessibility of logical records as related to the physical unit is the responsibility of the object program. In this manual, references to records means to logical records, unless the term 'physical record' is specifically used.

The concept of a logical record is not restricted to file data but is carried over into the definition of working storage and constants. Thus, working storage and constants may be grouped into logical records and defined by a series of record description entries.

3.3.1.3 Record Concepts

The Record Description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name, if required, followed by a series of independent clauses, as required.

3.3.2 CONCEPT OF LEVELS

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups, etc. Thus, an elementary item may belong to more than one group.

3.3.2.1 Level-Numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers, 66, 77 and 88, which are exceptions to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. The level-number of an item, either an elementary item or a group item, immediately following the last elementary item of the previous group, must be that of one of the groups to which the prior elementary item belongs.

Three types of entries exist for which there is no true concept of level. These are:

1. Entries that specify elementary items or groups introduced by a RENAMES clause,
2. Entries that specify noncontiguous Constant, Working-Storage, and Linkage data items,
3. Entries that specify condition-names.

Entries describing items by means of RENAMES clauses for the purpose of regrouping data items, have been assigned the special level-number 66.

Entries that specify noncontiguous data items, which are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level-number 77.

Entries that specify condition-names, to be associated with particular values of a conditional variable, have been assigned the special level-number 88.

3.3.2.2 Initial Values of Tables

In the Working-Storage and Constant sections, initial values of elements within tables are specified in one of the following ways:

1. The table may be described as a record by a set of contiguous data description entries, each of which specifies the VALUE of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (USAGE, PICTURE, etc.) may be used to complete the definition, where required. This form is required when the elements of the table require separate handling due to synchronization, USAGE, etc. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries, following the REDEFINES entry, which are repeated due to OCCURS clauses, must not contain VALUE clauses.

2. When the elements of the table do not require separate handling, the VALUE of the entire table may be given in the entry defining the entire table. The lower level entries will show the hierarchical structure of the table; lower level entries must not contain VALUE clauses.

3.3.3 CONCEPT OF CLASSES OF DATA

The five categories of data items (see 6.31, The PICTURE Clause) are grouped into three classes; alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric (without editing). Every elementary item belongs to one of the classes and further to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items subordinate to that group item. The following chart depicts the relationship of the class and categories of data items.

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

3.3.4 SELECTION OF CHARACTER REPRESENTATION AND RADIX

The value of a numeric item may be represented in either binary or decimal form depending on the equipment. In addition there are several ways of expressing decimal. Since these representations are actually combinations of bits, they are commonly called binary-coded decimal forms. The selection of radix is generally dependent upon the arithmetic capability of the computer. If more than one arithmetic radix is provided, the selection is dependent upon factors included in such clauses as USAGE and RECORDING MODE. The binary-coded decimal form is also used to represent characters and symbols that are alphanumeric items.

The selection of the proper binary-coded alphanumeric or binary-coded decimal form is dependent upon the capability of the computer and its external media.

When a computer provides more than one means of representing data, the Standard Data Format must be used if not otherwise specified by the data description. If both the external medium and the computer are capable of handling more than one form of data representation, or if there is no external medium associated with the data, the selection is dependent on factors included in USAGE, PICTURE, etc., clauses. Each implementor provides a complete explanation of the possible forms on the computer for which he is implementing COBOL. The method used in selecting the proper data form is also provided to allow the programmer to anticipate and/or control the selection.

The size of an elementary data item or a group item is the number of characters in Standard Data Format of the item. Synchronization and usage may cause a difference between this size and the actual number of characters required for the internal representation.

3.3.5 ALGEBRAIC SIGNS

Algebraic signs are used for two purposes: (1) To show whether the value of an item involved in an operation is positive or negative; and (2) To identify the value of an item as positive or negative on an edited report for external use.

Most forms of representation have a standard or normal manner of representing an operational sign. Thus, an indication that an operational sign is associated with an item is usually sufficient.

Since certain forms of representation allow alternative methods for representing operational signs, it is possible to describe certain types of operational signs which deviate from the normal method. Editing sign control characters are used to display the sign of an item and are not operational signs. These editing characters are only available through the use of the PICTURE clause.

3.3.6 ITEM ALIGNMENT FOR INCREASED OBJECT-CODE EFFICIENCY

Some computer memories are organized in such a way that there are natural addressing boundaries in the computer memory (e.g., word boundaries, half-word boundaries, byte boundaries). The way in which data is stored is determined by the object program, and need not respect these natural boundaries.

However, certain uses of data (e.g., in arithmetic operations or in subscripting) may be facilitated if the data is stored so as to be aligned on these natural boundaries. Specifically, additional machine operations in the object program may be required for the accessing and storage of data if portions of two or more data items appear between adjacent natural boundaries, or if certain natural boundaries bifurcate a single data item.

Data items which are aligned on these natural boundaries in such a way as to avoid such additional machine operations are defined to be synchronized. A synchronized item is assumed to be introduced and carried in that form; conversion to synchronized form occurs only during the execution of a procedure (other than READ or WRITE) which stores data in the item.

Synchronization can be accomplished in two ways:

a. By use of the SYNCHRONIZED clause,

b. By recognizing the appropriate natural boundaries and organizing the data suitably without the use of the SYNCHRONIZED clause.

Each implementor who provides for special types of alignment will specify the precise interpretations which are to be made.

3.3.7 UNIQUENESS OF DATA REFERENCE

3.3.7.1 Qualification

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because the name exists within a hierarchy of names such that the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and this process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. Within the Data Division, all data-names used for qualification must be associated with a level indicator or a level-number.

In the hierarchy of qualification, names associated with a level indicator are the most significant, then those names associated with level-number 01, then names associated with level-number 02, ... 49. A section-name is the highest (and the only) qualifier available for a paragraph-name. Thus, the most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and procedure-name.

Qualification is performed by following a data-name or a paragraph-name by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

The general formats for qualification are:

Format 1

$$\left\{\begin{array}{l}\text{data-name-1}\\ \text{condition-name}\end{array}\right\}\left[\left\{\begin{array}{l}\underline{\text{OF}}\\ \underline{\text{IN}}\end{array}\right\}\quad\text{data-name-2}\right]\ldots$$

Format 2

$$\text{paragraph-name}\quad\left[\left\{\begin{array}{l}\underline{\text{OF}}\\ \underline{\text{IN}}\end{array}\right\}\quad\text{section-name}\right]$$

The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

2. The same name must not appear at two levels in a hierarchy.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except REDEFINES where, by definition, qualification is unnecessary).

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same section.

5. A data-name cannot be subscripted when it is being used as a qualifier.

6. A name can be qualified even though it does not need qualifications; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.

3.3.7.2 Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names (see 6.29, The OCCURS Clause).

The subscript can be represented either by a numeric literal that is an integer, or by the special register TALLY, or by a data-name. The data-name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name may be qualified but not subscripted.

The subscript may contain a plus sign. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ... The highest permissible subscript value, in any particular case, is the maximum of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the terminal space of the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. Although not required, a comma may separate subscripts in a series. A space may appear between the left parenthesis and the left most subscript, and between the right parenthesis and the right most subscript.

The Format is:

$$\text{data-name (subscript} \left[\text{, subscript} \right] \ldots)$$

3.3.7.3 Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY clause in the definition of a table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index. An index-name must be initialized by a SET statement before it is used as a table reference (see 7.35, The SET Statement).

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the operator + or -, followed by an unsigned integral numeric literal all enclosed in the parentheses immediately after the terminal space of the data-name.

Data Description

The general format for indexing is:

$$\text{data-name} \quad (\text{index-name} \left[\left\{ \begin{matrix} + \\ - \end{matrix} \right\} \text{integer} \right] \left[\text{, index-name} \left[\left\{ \begin{matrix} + \\ - \end{matrix} \right\} \text{integer} \right] \right] \ldots)$$

3.3.7.4 Identifier

An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts or indices necessary to insure uniqueness.

The general formats for identifiers are:

Format 1

$$\text{data-name-1}\left[\begin{Bmatrix}\underline{\text{OF}}\\ \underline{\text{IN}}\end{Bmatrix}\ \text{data-name-2}\right]\ \ldots\ \left[(\text{subscript-1}\left[,\ \text{subscript-2}\left[,\ \text{subscript-3}\right]\right])\right]$$

Format 2

$$\text{data-name-1}\left[\begin{Bmatrix}\underline{\text{OF}}\\ \underline{\text{IN}}\end{Bmatrix}\ \text{data-name-2}\right]\ \ldots\ \left[(\text{index-name-1}\left[\begin{Bmatrix}+\\ -\end{Bmatrix}\text{integer}\right]\left[,\ \text{index-name-2}\left[\begin{Bmatrix}+\\ -\end{Bmatrix}\text{integer}\right]\left[,\ \text{index-name-3}\left[\begin{Bmatrix}+\\ -\end{Bmatrix}\text{integer}\right]\right]\right])\right]$$

Restrictions on qualification, subscripting and indexing are:

1. The commas shown in the general formats are not required.

2. A data-name must not itself be subscripted nor indexed when that data-name is being used as an index, subscript or qualifier.

3. Indexing is not permitted where subscripting is not permitted.

4. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data in a form specified by the implementor. Such data items are called index data items.

CHAPTER 4

IDENTIFICATION DIVISION

4.1 GENERAL DESCRIPTION

The Identification Division must be included in every COBOL source program. This division identifies both the source program and the resultant output listing. In addition, the user may include the date the program is written, the date the compilation of the source program is accomplished and such other information as desired under the paragraphs in the General Format shown below.

4.2 ORGANIZATION

Fixed paragraph names identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in the order of presentation shown by the General Format below.

4.2.1 STRUCTURE

The following format shows the structure of the Identification Division.

4.2.1.1 General Format

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name .

[AUTHOR. [comment-entry] . . .]

[INSTALLATION. [comment-entry] . . .]

[DATE-WRITTEN. [comment-entry]. . .]

[DATE-COMPILED. [comment-entry]. . .]

[SECURITY. [comment-entry]...]

ORGANIZATION

4.2.1.2 Syntax Rules

1. The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

4.2.1.3 General Rules

1. The comment-entry may be any combination of the characters from the computer's character set organized to conform to sentence and paragraph format.

   The following pages define the PROGRAM-ID paragraph and the DATE-COMPILED paragraph. While the other paragraphs are not defined, each General Format is formed in the same manner.

4.3 THE PROGRAM-ID PARAGRAPH

4.3.1 FUNCTION

The PROGRAM-ID paragraph gives the name by which a program is identified.

4.3.2 GENERAL FORMAT

PROGRAM-ID. program-name.

4.3.3 SYNTAX RULES

1. The program-name must conform to the rules for formation of a procedure-name. If the program-name is used to interact with the system, that portion of the name which is actually used is specified by the implementor.

4.3.4 GENERAL RULES

1. The PROGRAM-ID paragraph must contain the name of the program and must be present in every program.

2. The program-name identifies the source program and all listings pertaining to a particular program.

4.4 THE DATE-COMPILED PARAGRAPH

4.4.1 FUNCTION

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing.

4.4.2 GENERAL FORMAT

DATE-COMPILED. [comment-entry] . . .

4.4.3 GENERAL RULES

1. The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

   DATE-COMPILED. current date.

2. The comment-entry may be any combination of the characters from the computer's character set organized to conform to sentence and paragraph format.

CHAPTER 5

ENVIRONMENT DIVISION

5.1 GENERAL DESCRIPTION

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the compiling computer and the object computer. In addition, information relating to input-output control, special hardware characteristics and control techniques can be given.

The Environment Division must be included in every COBOL source program.

5.2 ORGANIZATION

Two sections make up the Environment Division: the Configuration Section and the Input-Output Section.

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run; and the SPECIAL-NAMES paragraph, which relates the implementor-names used by the compiler to the mnemonic-names used in the source program.

The Input-Output Section deals with the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph which names and associates the files with external media; and the I-O-CONTROL paragraph which defines special control techniques to be used in the object program.

5.2.1 STRUCTURE

The following is the general outline of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program.

5.2.1.1 General Format

```
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER.  source-computer-entry
  OBJECT-COMPUTER.  object-computer-entry
 [SPECIAL-NAMES.  special-names-entry]
 [INPUT-OUTPUT SECTION.
  FILE-CONTROL. {file-control-entry} ...
 [I-O-CONTROL.  input-output-control-entry]]
```

5.2.1.2 Syntax Rules

1. The Environment Division must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

2. The definitions of the entries for the contents of the paragraphs shown above are given on the following pages.

5.3 CONFIGURATION SECTION

5.3.1 THE SOURCE-COMPUTER PARAGRAPH

5.3.1.1 Function

The SOURCE-COMPUTER paragraph describes the computer upon which the program is to be compiled, and provides a means of communication with an executive routine.

5.3.1.2 General Format

Format 1

```
SOURCE-COMPUTER.  COPY library-name

[REPLACING { word-1       }  BY  { word-2       }
           { identifier-1 }      { identifier-2 }

        [, { word-3       }  BY  { word-4       } ] ... ] .
           { identifier-3 }      { identifier-4 }
```

Format 2

```
 SOURCE-COMPUTER.  computer-name  [WITH SUPERVISOR CONTROL ]
[                      (                       { WORDS      }                   ) ]
[   , MEMORY SIZE      (  integer              { CHARACTERS }                   ) ]
[                      (                       { MODULES    }                   ) ]
[                      (                                                        ) ]
[                      (  ADDRESS literal-1    { THRU    }   literal-2          ) ]
[                      (                       { THROUGH }                      ) ]
[                      (  [, literal-3 { THRU    }      literal-4] ...          ) ]
[                      (               { THROUGH }                              ) ]
   [, [literal-5]  implementor-name-1] ... .
```

SOURCE-COMPUTER

5.3.1.3 Syntax Rules

1. Computer-name is an implementor-name that must conform to the rules for formation of a procedure-name.

2. Each literal may be numeric or nonnumeric; when numeric, it must be an unsigned integer.

3. The words THRU and THROUGH are equivalent.

5.3.1.4 General Rules

1. For a discussion of the COPY function see Chapter 9, The COBOL Library.

2. Fixed computer-names and implementor-names are assigned by the individual implementor. Implementor-name may include input-output units, floating-point hardware, indicators such as breakpoints and sensing devices, index registers and any additional or special machine instructions, etc.

3. The computer-name may provide a means for describing equipment configuration, in which case the computer-name and its implied configuration are specified by each implementor. If the configuration implied by computer-name comprises more or less equipment than is actually needed by the compiler, the descriptive clauses following computer-name permit the specification of the actual subset of the configuration. The configuration definition contains specific information concerning the memory size, memory address, and quantity and types of hardware for a specific computer.

   The implementor defines what is to be done if the subset specified by the user is less than the minimum configuration required for compilation.

4. The SUPERVISOR option is used when the compiler is planned to be run under control of an operating system. This clause is for documentation purposes only, since the decision on mode of operating system control must have already been made prior to reading this entry in the source program. Communication between the compiler and the executive routine is specified by each implementor.

5.3.2 THE OBJECT-COMPUTER PARAGRAPH

5.3.2.1 Function

The OBJECT-COMPUTER paragraph describes the computer on which the program is to be executed, permits the specification of the input unit from which the computer instructions of the object program will be read at object time, and provides a means of communication with an executive routine.

5.3.2.2 General Format

Format 1

```
OBJECT-COMPUTER.  COPY library-name

[REPLACING {word-1      } BY {word-2      }
           {identifier-1}    {identifier-2}

          [, {word-3      } BY {word-4      } ] ... ].
             {identifier-3}    {identifier-4}
```

Format 2

```
OBJECT-COMPUTER.  computer-name  [WITH SUPERVISOR CONTROL]

 [                  {                  {WORDS     }                       }  ]
 [                  {  integer         {CHARACTERS}                       }  ]
 [                  {                  {MODULES   }                       }  ]
 [ , MEMORY SIZE    {                                                     }  ]
 [                  {  ADDRESS literal-1 {THRU   } literal-2              }  ]
 [                  {                    {THROUGH}                        }  ]
 [                  {   [, literal-3 {THRU   } literal-4] ...             }  ]
 [                  {                {THROUGH}                            }  ]

  [,      [literal-5]  implementor-name-1] ...

  [, SEGMENT-LIMIT IS priority-number]  [, ASSIGN OBJECT-PROGRAM TO

                                           input-unit].
```

OBJECT-COMPUTER

5.3.2.3 Syntax Rules

1. Computer-name is an implementor-name that must conform to the rules for formation of a procedure-name.

2. Each literal may be numeric or nonnumeric; when numeric, it must be an unsigned integer.

3. Priority-number must be an integer ranging in value from 1 through 49.

4. The words THRU and THROUGH are equivalent.

5.3.2.4 General Rules

1. For a discussion of the COPY function see Chapter 9, The COBOL Library.

2. The computer-name may provide a means for describing equipment configuration, in which case the computer-name and its implied configuration are specified by each implementor. If the configuration implied by computer-name comprises more or less equipment than is actually needed by the object program, the descriptive clauses following computer-name permit the specification of the actual subset of the configuration. The configuration definition contains specific information concerning the memory size, memory address, and quantity and types of hardware for a specific computer.

   The implementor defines what is to be done if the subset specified by the user is less than the minimum configuration required for running the object program.

3. The SUPERVISOR clause is used when the object program is to be executed under the control of an executive routine. Communication between the object program and the executive routine is specified by the implementor.

4. Implementor-name may include input-output units, floating-point hardware, indicators such as breakpoints and sensing devices, index registers and any additional or special machine instructions, etc.

5. The optional SEGMENT-LIMIT clause gives the limit for segmentation to be varied by the user depending upon the configuration specified in the OBJECT-COMPUTER paragraph.

6. If the ASSIGN OBJECT-PROGRAM clause is used, the compiler assigns the specified input unit as the unit from which the object program computer instructions are read at object time. The names for the allowable input units are specified by the implementor. If the clause is omitted, the compiler assigns the implementor-defined standard input unit for this purpose.

5.3.3 THE SPECIAL-NAMES PARAGRAPH

5.3.3.1 Function

The SPECIAL-NAMES paragraph provides a means of relating implementor-names to user-specified mnemonic-names.

5.3.3.2 General Format

Format 1

```
SPECIAL-NAMES.  COPY library-name

[REPLACING {word-1      } BY {word-2      }
           {identifier-1}    {identifier-2}

        [, {word-3      } BY {word-4      } ]...].
           {identifier-3}    {identifier-4}
```

Format 2

```
                                      { IS mnemonic-name [, ON STATUS
                                      {
                                      { IS mnemonic-name [, OFF STATUS
SPECIAL-NAMES.     [implementor-name  {
                                      { ON STATUS IS condition-name-1
                                      {
                                      { OFF STATUS IS condition-name-2

    IS condition-name-1 [, OFF STATUS IS condition-name-2]]  }
    IS condition-name-2 [, ON STATUS IS condition-name-1]]   } ] ...
    [, OFF STATUS IS condition-name-2]                       }
    [, ON STATUS IS condition-name-1]                        }
    [, CURRENCY SIGN IS literal]  [, DECIMAL-POINT IS COMMA].
```

5.3.3.3 Syntax Rules

1. The SPECIAL-NAMES paragraph is required if mnemonic-names, condition-names, the DECIMAL-POINT clause or the CURRENCY SIGN clause are used.

2. In repetition, a comma may be used before implementor-name.

SPECIAL-NAMES

5.3.3.4 General Rules

1. For a discussion of the COPY function see Chapter 9, The COBOL Library.

2. If the implementor-name is not a switch, the associated mnemonic-name may be used in the ACCEPT, DISPLAY, SUSPEND and WRITE statements and in the CODE and LINAGE clauses.

3. If the implementor-name is a switch, at least one condition-name must be associated with it. The status of the switch is specified by condition-names and interrogated by testing the condition-names (see 7.4.5, Switch-Status Condition).

4. The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters:

   a. digits '0' thru '9';

   b. alphabetic characters 'A', 'B', 'C', 'D', 'J', 'K', 'L', 'P', 'R', 'S', 'V', 'X', 'Z' or the space;

   c. special characters '*', '+','-', ',', '.', ';', '(', ')', '"'.

   If this clause is not present, only the currency sign is used in the PICTURE clause.

5. The clause DECIMAL-POINT IS COMMA means that the function of comma and period are exchanged in the PICTURE clause character-string and in numeric literals.

5.4 INPUT-OUTPUT SECTION

5.4.1 THE FILE-CONTROL PARAGRAPH

5.4.1.1 Function

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows particular hardware assignments.

5.4.1.2 General Format

Format 1

```
FILE-CONTROL.  COPY  library-name

[REPLACING {word-1      } BY {word-2      }
           {identifier-1}    {identifier-2}

        [, {word-3      } BY {word-4      } ] ... ] .
           {identifier-3}    {identifier-4}
```

Format 2

```
FILE-CONTROL. { SELECT [OPTIONAL] file-name

ASSIGN TO  [integer-1]  implementor-name-1  [, implementor-name-2] ...

[FOR MULTIPLE {REEL}]  [, RESERVE {integer-2} ALTERNATE [AREA ] ]
              {UNIT}              {NO       }           [AREAS]

[, PRIORITY IS implementor-name-3]

[ , {FILE-LIMIT IS   }  {data-name-1}  {THRU   }  {data-name-2}
    {FILE-LIMITS ARE }  {literal-1  }  {THROUGH}  {literal-2  }

      [, {data-name-3}  {THRU   }  {data-name-4} ]  ... ]
         {literal-3  }  {THROUGH}  {literal-4  }

[ , ACCESS MODE IS      {SEQUENTIAL} ]
                        {RANDOM    }

[ , PROCESSING MODE IS  {SEQUENTIAL                          } ]
                        {RANDOM FOR integer-3 RECORDS        }

[ , ACTUAL KEY IS data-name-5] .   } ...
```

Format 3

```
FILE-CONTROL.  {SELECT  [OPTIONAL]  file-name

ASSIGN TO implementor-name-4

[, implementor-name-5] ... OR implementor-name-6 [, implementor-name-7] ...

[FOR MULTIPLE {REEL}]  [, RESERVE {integer-4}  ALTERNATE [AREA ] ]
              {UNIT}              {NO       }            [AREAS]

[, PRIORITY IS implementor-name-8 ].}  ...
```

5.4.1.3 Syntax Rules

1. Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph following the key word SELECT. Each selected file must have a File Description entry in the Data Division.

2. The OPTIONAL clause is allowed—only for input files accessed in a sequential manner. It is required for sequential input files that are not necessarily present each time the object program is executed.

3. Integer-1 indicates the number of input-output units of a given medium assigned to the file-name. If an integer is not specified, the compiler determines the number of units to be assigned.

4. In Format 2, if file-name is a sort-file, only the ASSIGN clause and the PRIORITY clause are permitted to follow the file-name in the FILE-CONTROL paragraph.

5. The ACCESS MODE and PROCESSING MODE clauses must be given for mass storage files.

6. Any restrictions on the COBOL description of the operands in the ACTUAL KEY and FILE-LIMIT clauses are specified by the implementor.

7. The words THRU and THROUGH are equivalent.

8. Format 3 can only be used with the GIVING option of a SORT statement and specifies the final output file.

9. The FILE-CONTROL paragraph is required when the Input-Output Section header is present.

5.4.1.4 General Rules

1. For a discussion of the COPY function see Chapter 9, The COBOL Library.

2. All files used in the program must be assigned to an input or output medium (implementor-name). The implementor-name is specified by the implementor for the specific hardware unit.

3. The MULTIPLE REEL clause must be included whenever the number of tape units assigned, explicitly or implicitly, might be less than the number of reels in the file. The MULTIPLE UNIT clause must be included whenever the number of mass storage devices assigned might be less than the number of mass storage units in the file.

4. The RESERVE clause allows the user to modify the number of input-output areas allocated by the compiler. The option RESERVE integer ALTERNATE AREAS means that integer additional areas are to be reserved for the file in addition to the minimum area. The implementor specifies what is the minimum area and what is the additional area for particular hardware.

5. When specific input or output units are assigned by the user, these units must be assigned to all files existing on the same reel for reel-oriented storage media (see 5.4.2, The I-O-CONTROL Paragraph).

6. The PRIORITY clause provides a means of assigning priorities to individual files for multiprogramming operations. The manner in which the PRIORITY is specified is defined by the implementor.

7. The FILE-LIMIT clause specifies that:

    a. For Sequential Access, logical records are obtained or placed sequentially in the mass storage file by the implicit progression from segment to segment. The AT END imperative-statement of a READ statement is executed when the logical end of the last segment of the file is reached and an attempt is made to read that file. The INVALID KEY clause of a WRITE statement is executed when the end of the last segment is reached and an attempt is made to write on that file.

    b. For Random Access, logical records are obtained or placed randomly in the mass storage file within these file limits. The contents of the ACTUAL KEY data items that are not within these limits cause the execution of the INVALID KEY clause on the READ and the WRITE statements.

8. In the FILE-LIMIT clause, each pair of operands associated with the key word THRU represents a logical segment of the file. The logical beginning of a mass storage file segment is considered to be that address represented by the first operand of the pair of the FILE-LIMIT clause; the logical end of a mass storage file segment is considered to be that address as represented by the last operand of the pair of the FILE-LIMIT clause.

9. File limit information may also be specified in the ASSIGN clause. If this information is provided in both places, the value of the data item as specified in the FILE-LIMIT clause must be within the range of the limits specified in the ASSIGN clause.

10. The value of the data items as specified in the FILE-LIMIT clause is utilized by the Mass Storage Control System only at the time that the applicable mass storage file is opened by the execution of the OPEN statement.

11. For the ACCESS MODE SEQUENTIAL clause, the mass storage records are obtained or placed sequentially. That is, the next logical record is made available from the file on a READ statement execution or a specific logical record is placed into the file on a WRITE statement execution. No ACTUAL KEY entries are necessary for the SEQUENTIAL mode.

12. If the ACCESS MODE RANDOM clause is specified, the ACTUAL KEY entry must also be specified. In this case, the Mass Storage Control System obtains each record randomly. That is, the specified logical record (located using ACTUAL KEY data-name contents) is made available from the file on a READ statement execution or is placed in a specific location on the file (using ACTUAL KEY data-name contents) on a WRITE statement execution.

13. For the PROCESSING MODE SEQUENTIAL clause, the mass storage records are processed in the order in which they are accessed.

14. For the PROCESSING MODE RANDOM clause, the mass storage records are processed in an asynchronous, random manner, without regard for the order in which they are accessed. However, the PROCESSING MODE RANDOM clause may be used only in conjunction with the ACCESS MODE RANDOM clause. The asynchronous, random processing of the mass storage records requires that the user write the associated procedural statements in the USE entry in a Declarative Section and that the user refer to these procedural statements in the Procedure Division by the PROCESS statement.

15. The FOR integer-3 RECORDS specification indicates the number of mass storage record areas which are provided to hold mass storage records for asynchronous processing, and thus indicates the maximum number of asynchronous processing cycles which are possible for that mass storage file. If integer-3 is specified as 1, no asynchronous processing is possible for the mass storage file and, in effect, the specification becomes equivalent to the clause PROCESSING MODE IS SEQUENTIAL.

16. Although the ACTUAL KEY clause is optional for Sequential Access (the Mass Storage Control System maintains an internal key), if the ACTUAL KEY clause is specified, the Mass Storage Control System updates data-name-5 by the following rules.

    a. Following a WRITE statement execution, the contents of the ACTUAL KEY data-item are always implicitly updated.

    b. Prior to a READ statement execution, the contents of the ACTUAL KEY data item are implicitly updated only if not logically preceded by a WRITE statement execution.

    Since the implicit updating of the ACTUAL KEY data item is a function of the implementor's Mass Storage Control System and the specification is never referred to or required by the Mass Storage Control System, any change the programmer may make to the ACTUAL KEY data item does not affect the mass storage file processing but may result in unpredictable values if subsequent reference is made by the programmer to the contents of the ACTUAL KEY data item.

17. If the access mode for this file is specified as RANDOM, the ACTUAL KEY clause must be provided; the contents of data-name-5 is used by the SEEK statement (or, in its absence, the READ and WRITE statements) to locate a specific mass storage record. Thus, the actual location (address) must have been placed in data-name-5 prior to the execution of the SEEK statement (or the implicit SEEK associated with the READ and WRITE statements).

18. The presence of the OR clause means the file referenced by file-name emerges from the sorting operation either on the assigned hardware units preceding the key word OR, or on the hardware units following the key word OR. At the conclusion of the sorting operation, an indication will be given of which hardware units contain the file. The proper hardware units are addressed when this file is opened for input.

5.4.2 THE I-O-CONTROL PARAGRAPH

5.4.2.1 Function

The I-O-CONTROL paragraph specifies the input-output techniques, the points at which rerun is to be established, the memory area which is to be shared by different files, and the location of files on a multiple-file reel.

5.4.2.2 General Format

Format 1

```
I-O-CONTROL.  COPY library-name

[REPLACING  {word-1      } BY {word-2      }
            {identifier-1}    {identifier-2}

       [, {word-3      } BY {word-4      } ] ...].
          {identifier-3}    {identifier-4}
```

Format 2

```
I-O-CONTROL.  [APPLY input-output-technique ON file-name-1

[, file-name-2] ...]...

[; RERUN [ON      {file-name-3     } ]
                  {implementor-name}
              (  {[END  OF] {REEL}  }                   )
              (  {          {UNIT}  }  OF file-name-4   )
              (  { integer-1 RECORDS}                   )
     EVERY    (                                         )  ...]
              (    integer-2 CLOCK-UNITS                )
              (                                         )
              (    condition-name                       )

[; SAME [{RECORD}] AREA FOR file-name-5 {, file-name 6} ...]...
         {SORT  }

[; MULTIPLE FILE TAPE CONTAINS file-name-8  [POSITION

     integer-3] [, file-name-9 [POSITION integer-4]] ...]... .
```

5.4.2.3 Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. File-names in one APPLY clause must not be used in another APPLY clause.

3. APPLY clauses may be separated by the semicolon.

4. A file-name that represents a sort-file cannot appear in a RERUN or MULTIPLE FILE option.

5. The END OF UNIT option may only be used if file-name-4 is a sequentially-accessed mass storage file. The definition of UNIT is determined by each implementor.

6. When either the integer-1 RECORDS option or the integer-2 CLOCK-UNITS option is specified, implementor-name must be given in the RERUN option.

7. A file-name that represents a sort-file must not appear in the SAME clause unless the SORT or RECORD option is used.

8. The three forms of the SAME clause (SAME AREA, SAME RECORD AREA, SAME SORT AREA) are considered separately in the following:

   More than one SAME clause may be included in a program, however:

   a. A file-name must not appear in more than one SAME AREA clause.

   b. A file-name must not appear in more than one SAME RECORD AREA clause.

   c. A file-name that represents a sort-file must not appear in more than one SAME SORT AREA clause.

   d. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

   e. If a file-name that does not represent a sort-file appears in a SAME AREA clause and one or more SAME SORT AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA clause(s).

5.4.2.4 General Rules

1. For a discussion of the COPY function see Chapter 9, The COBOL Library.

2. When the implementor furnishes more than one input-output system technique, the APPLY input-output technique option permits the user to select the appropriate technique for his object program. Specific names to designate the individual input-output techniques are specified by the implementor.

   The implementor must provide at least one of the specified forms of the RERUN clause.

   The RERUN clause specifies where the rerun information is recorded and when the memory dump occurs. Memory dumps are recorded in the following ways:

   a. The memory dump is written on each reel or unit of an output file and the implementor specifies where, on the reel or file, the memory dump is to be recorded;

   b. The memory dump is written on a separate rerun tape or unit, as specified by the implementor-name given in the RERUN option.

5. Listed below are several conditions under which rerun points can be established:

   a. When either the END OF REEL or END OF UNIT option is used and it is also desired to write the memory dump on an output file, as specified by file-name-4. In this case, file-name-3 is not required.

   b. When file-name-3, which must be an output file, is specified in the RERUN option, normal reel- or unit-closing functions for file-name-3 are performed along with the memory dump. In this case, file-name-4 may either be an input file or an output file.

   c. When a number of records, specified by integer-1, of an input or an output file, file-name-4, have been processed. In this case, implementor-name must be specified.

   d. When an interval of time, specified by the use of the CLOCK-UNITS option, calculated by an internal clock, has elapsed. In this case, implementor-name must be specified.

   e. When a hardware switch assumes a particular status as specified by condition-name. In this case, the associated hardware switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

6. The SAME AREA clause specifies that two or more files that do not represent sort-files are to use the same memory area during processing. The area being shared includes all storage areas (including alternate areas) assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time. (See 5.4.2.3, Syntax Rule 8.d.)

7. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause.

8. If the SAME SORT AREA clause is used, at least one of the file-names must represent a sort-file. Files that do not represent sort-files may also be named in the clause. This clause specifies that storage is shared as follows:

   a. The SAME SORT AREA clause specifies a memory area which will be made available for use in sorting each sort-file named. Thus any memory area allocated for the sorting of a sort-file is available for re-use in sorting any of the other sort-files.

   b. In addition, storage areas assigned to files that do not represent sort-files may be allocated as needed for sorting the sort-files named in the SAME SORT AREA clause. The extent of such allocation will be specified by the implementor.

   c. Files other than sort-files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, he must also include in the program a SAME AREA or SAME RECORD AREA clause naming these files.

   d. During the execution of a SORT statement that refers to a sort-file named in this clause, any non-sort-files named in this clause must not be open.

9. The MULTIPLE FILE option is required when more than one file shares the same physical reel of tape. Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION option need not be given. If any file in the sequence is not listed, the position relative to the beginning of the tape must be given. Not more than one file on the same tape reel may be open at one time.

CHAPTER 6

THE DATA DIVISION

6.1 GENERAL DESCRIPTION

6.1.1 OVERALL APPROACH

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed falls into three categories:

a. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.

b. That which is developed internally and placed into intermediate or working-storage, or placed into specific format for output reporting purposes.

c. Constants which are defined by the user.

6.1.2 PHYSICAL AND LOGICAL ASPECTS OF DATA DESCRIPTION

6.1.2.1 Data Division Organization

The Data Division, which is one of the required divisions in a program, is subdivided into sections. These are the File, Working-Storage, Constant, Linkage, and Report Sections.

The File Section defines the contents of data files stored on an external medium. Each file is defined by a file description followed by a record description or a series of record descriptions. When the file description specifies a file to be used only as a Report Writer output file, the record description can be omitted. The Working-Storage Section describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. The Constant Section describes data items whose values are assigned in the source program and do not change during the execution of the object program. Like the Working-Storage Section, the Constant Section may specify both logical records and noncontiguous items. The Linkage Section appears in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the Working-Storage and Constant Sections. The Report Section describes the content and format of reports that are to be generated.

6.1.2.2 Data Division Structure

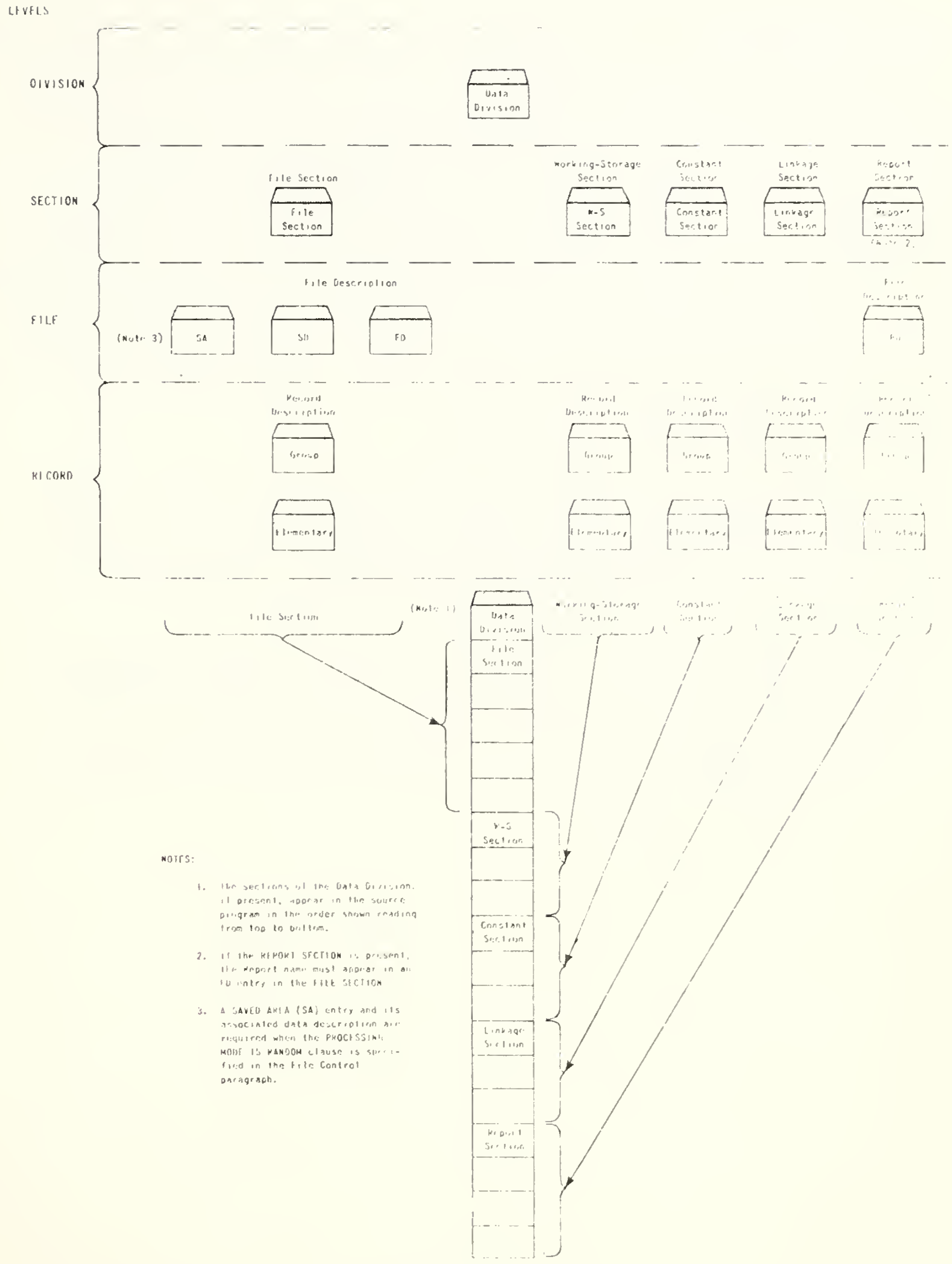
The Data Division is prepared according to the reference format described in Chapter 10. The Data Division is identified by and must begin with the following header:

<u>DATA</u> <u>DIVISION</u> [<u>PREPARED</u> FOR computer-name].

The optional clause PREPARED FOR computer-name, is used only when the data descriptions have been written for a computer other than the object computer. Each of the sections of the Data Division is optional and may be omitted from the source program if not needed. The fixed names of these sections in their required order of appearance as section headers in the Data Division are as follows:

```
FILE SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.
LINKAGE SECTION.
REPORT SECTION.
```

The section headers for the File Section and the Report Section are followed by one or more sets of entries composed of file clauses, followed by associated record description entries. Working-Storage, Constant and Linkage Section headers are followed by data description entries for noncontiguous items, followed by record descriptions. This is illustrated in Figure 6-1, Data Division Structure.



NOTES:

1. The sections of the Data Division, if present, appear in the source program in the order shown reading from top to bottom.

2. If the REPORT SECTION is present, the Report name must appear in an FD entry in the FILE SECTION

3. A SAVED AREA (SA) entry and its associated data description are required when the PROCESSING MODE IS RANDOM clause is specified in the File Control paragraph.

Figure 6-1 DATA DIVISION STRUCTURE

6.2 FILE SECTION

In a COBOL program the File Description entries (FD, SD) represent the highest level of organization in the File Section. The Sort File Description (SD) is a special type of file description. The File Section has a special provision for reserving memory for mass storage processing via a Saved-Area (SA) entry and subordinate descriptions. The File Section header is followed by a File Description entry consisting of a level indicator (FD, or SD), a data-name and a series of independent clauses. These clauses specify the manner in which the data is recorded on the file, the size of the logical and physical records, the names of the label records contained in the file and values of label items, the names of the data records and reports which comprise the file and finally, the keys on which the data records are sequenced. The entry itself is terminated by a period.

An SD File Description gives information about the name, size, and number of data records in the sort file. The name Sort File designates a set of records to be sorted by a SORT statement. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT statement.

When the processing mode for a file is random, an SA entry must appear in the File Section, followed by record description entries for data referred to by the out-of-line procedures. The SA entry, like the BLOCK CONTAINS clause in the FD entry, describes a memory area containing a number of records, each of which may be of any of the records described following the SA entry.

6.3 RECORD DESCRIPTION-STRUCTURE

A Record Description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A Record Description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in 3.3.2, Concepts of Levels while the elements allowed in a Record Description are shown in 6.12, the Data Description Skeleton.

6.4 WORKING-STORAGE SECTION

The Working-Storage Section is composed of the section header, followed by data description entries for noncontiguous Working-Storage items and record description entries in that order. Each Working-Storage Section record name and noncontiguous item name must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

6.4.1 NONCONTIGUOUS WORKING-STORAGE

Items in Working-Storage which bear no hierarchic relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number, 77.

The following data clauses are required in each data description entry:

a. level-number 77

b. data-name

c. the PICTURE clause.

Other data description clauses are optional and can be used to complete the description of the item if necessary.

6.4.2 WORKING-STORAGE RECORDS

Data elements in Working-Storage which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of Record Descriptions. All clauses which are used in normal input or output Record Descriptions can be used in a Working-Storage Record Description.

6.4.3 INITIAL VALUES

The initial value of any item in the Working-Storage Section except an index data item is specified by using the VALUE clause with the data item. The initial value of any index data-item is unpredictable.

The skeletal format for the Working-Storage Section is as follows:

```
WORKING-STORAGE SECTION.
  77 data-description entry
    88 condition-name-1
      .
      .
      .
  77 data-description entry
  01 data-description entry
     02 data-description entry
      .
      .
      .
  66 data-name-n RENAMES data-name-m
  01 data-description entry
     02 data-description entry
        03 data-description entry
           88 condition-name-2
```

6.5 CONSTANT SECTION

Constant storage is that part of computer memory set aside to save named constants for use in a given program. The concept of literals and figurative constants enables the user to specify the value of a constant by writing its actual value or a figurative representation of that value.

The Constant Section is organized in exactly the same way as the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous constants, and then by data description entries for contiguous constant records in that order. Each Constant Section record-name and noncontiguous item name must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

6.5.1 NONCONTIGUOUS CONSTANT STORAGE

Constants that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these constants is defined in a separate data description entry which begins with the special level-number 77.

The following data clauses are required in each data description entry:

a. level-number 77
b. data-name
c. the PICTURE clause
d. VALUE.

The clause USAGE IS INDEX must not be used in the entry. Other data clauses are optional and can be used to complete the description of the constant when necessary.

6.5.2 CONSTANT RECORDS

Constants that bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of Record Descriptions.

All data clauses, except USAGE IS INDEX, can be used in a Constant Record Description.

6.5.3 VALUE OF CONSTANTS

The value of every item in the Constant Section must be specified by a VALUE clause, stated in the elementary item entry or in the group item entry.

The skeletal format for the Constant Section is as follows:

```
CONSTANT SECTION.
 77 data-description entry
    .
    .
    .
 77 data-description entry
 01 data-description entry
    02 data-description entry
    .
    .
    .
 01 data-description entry
    02 data-description entry
       03 data-description entry
```

6.6 LINKAGE SECTION

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. Procedure Division references to a data item in the Linkage Section are resolved at object-time by equating the reference in the called program to the location used in the calling program. The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous Linkage items and record description entries in that order. Each Linkage Section record-name and noncontiguous item name must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification. Every data item described in the Linkage Section of the called program must have been defined in the File, Working-Storage, Constant, or Linkage Section of the calling program.

6.6.1 NONCONTIGUOUS LINKAGE STORAGE

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these data items is defined in a separate data description entry which begins with the special level-number 77.

The following data clauses are required in each data description entry:

a. level-number 77
b. data-name
c. the PICTURE clause.

Other data description clauses are optional and can be used to complete the description of the item if necessary.

6.6.2 LINKAGE RECORDS

Data elements in the Linkage Section which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of Record Description. All clauses which are used in normal input or output Record Descriptions can be used in a Linkage Record Description.

6.6.3 INITIAL VALUES

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

The skeletal format for the Linkage Section is as follows:

```
LINKAGE SECTION.
 77 data-description entry
    88 condition-name-1
    .
    .
    .
 77 data-description entry
 01 data-description entry
    02 data-description entry
    .
    .
    .
 66 data - name-n RENAMES data-name-m
 01 data-description entry
    02 data-description entry
       03 data-description entry
          88 condition-name-2
```

6.7 REPORT SECTION

The Report Section consists of two types of entries for each report; one describes the physical aspects of the report format, the other type describes conceptual characteristics of the items which make up the report and their relation to the report format. These are:

a. Report Description entry RD;

b. Report Group Description entries.

6.7.1 REPORT DESCRIPTION ENTRY

The Report Description entry contains information pertaining to the overall format of a report named in the File Section and is uniquely identified in the Report Section by the level indicator RD. The characteristics of the report page are provided by describing the number of physical lines per page and the limits for presenting specified headings, footings, and details within a page structure. Data items which act as control factors during presentation of the report are specified in the RD entry. Each report named in an FD entry in the File Section must be defined by an RD entry.

6.7.2 REPORT GROUP DESCRIPTION ENTRY

A report may be divided into report groups. A report group is a set of data items that is to be presented as an individual unit, irrespective of its physical format structure. It may consist of several report lines containing many data items or of one report line containing a single data item. Three categories of report group definitions are provided: heading groups, footing groups and detail groups.

The data items constituting a report group must be identified by the level-number 01 and a TYPE clause. Report group names are required when reference is made in the Procedure Division:

a. to a TYPE DETAIL report group by a GENERATE statement

b. to a TYPE HEADING or FOOTING report group by a USE statement.

The description of the report group, analogous to that of the data record, consists of a set of entries defining the characteristics of the elements. The placement of an item in relation to the entire report group and to the overall report format, the format description of all items, and any control factors associated with the group are defined by the entry.

6.8 THE FILE DESCRIPTION--COMPLETE ENTRY SKELETON

6.8.1 FUNCTION

The File Description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

6.8.2 GENERAL FORMAT

Format 1

```
FD file-name; COPY library-name

 [ REPLACING { word-1       } BY { word-2       }
 [           { identifier-1 }    { identifier-2 }

          [ , { word-3       } BY { word-4       } ]     ]
          [   { identifier-3 }    { identifier-4 } ] ... ] .
```

Format 2

```
 FD file-name

  [; RECORDING MODE IS mode-name   ]

  [; BLOCK CONTAINS  [integer-1 TO]  integer-2 { RECORDS    } ]
  [                                            { CHARACTERS } ]

  [; RECORD CONTAINS [integer-3 TO]  integer-4   CHARACTERS]

   ; LABEL { RECORDS ARE } { STANDARD                                  }
           { RECORD IS   } { OMITTED                                   }
                           { data-name-1   [, data-name-2]   ...       }

 [ ; VALUE OF data-name-3 IS { data-name-4   }
 [                           { literal-1     }
       [   , data-name-5 IS  { data-name-6   } ]    ]
       [                     { literal-2     } ] ...]

 [ ; DATA { RECORD IS   }  data-name-7   [, data-name-8]  ... ]
 [        { RECORDS ARE }                                     ]
```

Format 2 (Cont.)

$$\left[;\left\{\begin{array}{l}\underline{\text{REPORT}}\text{ IS}\\ \underline{\text{REPORTS}}\text{ ARE}\end{array}\right\}\text{report-name-1}\quad [\text{, report-name-2}]\quad \ldots \right]$$

$$\left[;\ \underline{\text{LINAGE}}\text{ IS}\quad \left\{\begin{array}{l}\text{identifier-1 }\underline{\text{LINES}}\\ \text{integer-5 }\underline{\text{LINES}}\\ \text{mnemonic-name}\end{array}\right\} \right]$$

6.8.3 SYNTAX RULES

1. The level indicator FD identifies the beginning of a File Description and must precede the file-name.

2. All semicolons are optional in the File Description but the entry must be terminated by a period.

3. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial.

6.8.4 GENERAL RULES

1. Format 1 is used when the COBOL library contains the File Description entry, otherwise Format 2 is used.

2. For a discussion of the COPY function see Chapter 9, The COBOL Library.

6.9 THE SORT FILE DESCRIPTION--COMPLETE ENTRY SKELETON

6.9.1 FUNCTION

The Sort File Description furnishes information concerning the physical structure, identification, and record names of the file to be sorted.

6.9.2 GENERAL FORMAT

Format 1

$$\underline{\text{SD}}\ \text{file-name};\ \underline{\text{COPY}}\ \text{library-name}$$

$$\left[\underline{\text{REPLACING}}\begin{Bmatrix}\text{word-1}\\ \text{identifier-1}\end{Bmatrix}\underline{\text{BY}}\begin{Bmatrix}\text{word-2}\\ \text{identifier-2}\end{Bmatrix}\left[\ ,\begin{Bmatrix}\text{word-3}\\ \text{identifier-3}\end{Bmatrix}\underline{\text{BY}}\begin{Bmatrix}\text{word-4}\\ \text{identifier-4}\end{Bmatrix}\right]\ldots\right].$$

Format 2

$$\underline{\text{SD}}\ \text{file-name}$$

$$[\ ;\ \underline{\text{RECORD}}\ \text{CONTAINS}\ \ [\text{integer-1}\ \underline{\text{TO}}]\ \ \text{integer-2 CHARACTERS}]$$

$$[\ ;\ \underline{\text{DATA}}\begin{Bmatrix}\underline{\text{RECORD}}\ \text{IS}\\ \underline{\text{RECORDS}}\ \text{ARE}\end{Bmatrix}\ \text{data-name-1}\ \ [\ ,\ \text{data-name-2}\ ]\ \ldots].$$

6.9.3 SYNTAX RULES

1. The level indicator SD identifies the beginning of the Sort File Description and must precede the file-name.

2. All semicolons are optional in the Sort File Description but the entry must be terminated by a period.

3. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial.

6.9.4 GENERAL RULES

1. Format 1 is used when the COBOL library contains the Sort File Description entry, otherwise Format 2 is used.

2. For a discussion of the COPY function see Chapter 9, The COBOL Library.

6.10 THE SAVED AREA DESCRIPTION--COMPLETE ENTRY SKELETON

6.10.1 FUNCTION

The Saved Area Description furnishes information concerning the physical structure pertaining to the saved area required for random processing.

6.10.2 GENERAL FORMAT

Format 1

$$\underline{\text{SA}}\ \text{area-name};\ \underline{\text{COPY}}\ \text{library-name}$$

$$\left[\underline{\text{REPLACING}}\left\{\begin{array}{l}\text{word-1}\\ \text{identifier-1}\end{array}\right\}\underline{\text{BY}}\left\{\begin{array}{l}\text{word-2}\\ \text{identifier-2}\end{array}\right\}\right.$$

$$\left.\left[,\left\{\begin{array}{l}\text{word-3}\\ \text{identifier-3}\end{array}\right\}\underline{\text{BY}}\left\{\begin{array}{l}\text{word-4}\\ \text{identifier-4}\end{array}\right\}\right]\ \ldots\right].$$

Format 2

$$\underline{\text{SA}}\ \text{area-name};\ \underline{\text{AREA}}\ \text{CONTAINS integer-1}\ \left\{\begin{array}{l}\text{CHARACTERS}\\ \underline{\text{RECORDS}}\end{array}\right\}$$

$$\left[;\ \underline{\text{RECORD}}\ \text{CONTAINS}\ \left[\text{integer-2}\ \underline{\text{TO}}\right]\ \text{integer-3 CHARACTERS}\right].$$

6.10.3 SYNTAX RULES

1. An SA entry is required when random processing is specified.
2. A level indicator SA is used to identify the beginning of a Saved Area Description entry and must precede the unique area-name.

3. Semicolons are optional in the SA entry but the entry must be terminated by a period.

4. The order in which the clauses appear is immaterial.

6.10.4 GENERAL RULES

1. Format 1 is used when the COBOL library contains the Saved Area Description entry, otherwise Format 2 is used.

2. For a discussion of the COPY function see Chapter 9, The COBOL Library.

6.11 THE REPORT DESCRIPTION--COMPLETE ENTRY SKELETON

6.11.1 FUNCTION

The Report Description furnishes information concerning the physical structure and identification for a particular named report.

6.11.2 GENERAL FORMAT

Format 1

```
RD report-name; COPY library-name

[ REPLACING { word-1         } BY { word-2         }
  .         { identifier-1   }    { identifier-2   }

        [ , { word-3         } BY { word-4         } ] ... ] .
            { identifier-3   }    { identifier-4   }
```

Format 2

```
RD report-name
[;CODE mnemonic-name-1]

[ ; { CONTROL IS   } { FINAL                                              } ]
    { CONTROLS ARE } { identifier-1  [, identifier-2] ...                 }
                     { FINAL, identifier-1  [, identifier-2] ...          }

[ ; PAGE   { LIMIT IS   }  integer-1  { LINE  }  [, HEADING integer-2]
           { LIMITS ARE }             { LINES }

        [, FIRST DETAIL  integer-3]  [, LAST DETAIL integer-4]

        [, FOOTING integer-5] ] .
```

6.11.3 SYNTAX RULES

1. The level indicator RD identifies the beginning of a Report Description and must precede the report-name.

2. The report-name must appear in at least one FD entry REPORT clause.

3. All semicolons are optional in the Report Description but the entry must be terminated by a period.

4. The clauses which follow the report-name are optional in many cases, and, except in Format 1, their order of appearance is immaterial.

6.11.4 GENERAL RULES

1. Format 1 is used when the COBOL library contains the Report Description entry, otherwise Format 2 is used. If the library-name is not unique, it may be qualified by indicating the file-identification in the OF program-file clause.

2. For a discussion of the COPY function see Chapter 9, The COBOL Library.

3. The fixed data-names, LINE-COUNTER and PAGE-COUNTER, are automatically generated by the Report Writer based on the presence of specific entries and are not data clauses. The descriptions of these two counters are included here in order to explain their resultant effect on the over-all report format.

6.11.5 PAGE-COUNTER RULES

1. A PAGE-COUNTER is a counter generated by the Report Writer to be used as a SOURCE data item in order to automatically present consecutive page numbers.

2. One PAGE-COUNTER is supplied for each report described in the Report Section. The numeric counter is based on the size specified in the PICTURE clause associated with the elementary SOURCE data item description.

3. If more than one PAGE-COUNTER is given as a SOURCE data item within a given report the number of numeric characters indicated by the PICTURE clauses must be identical. The size must indicate sufficient numeric character positions to prevent overflow.

4. If more than one Report Description entry exists in the Report Section, the user must qualify PAGE-COUNTER by the report-name. PAGE-COUNTER may be referred to in Data Division clauses and by Procedure Division statements.

5. PAGE-COUNTER is automatically set to 1 (one) initially by the Report Writer; if a starting value for PAGE-COUNTER other than 1 (one) is desired, the programmer may change the contents of the PAGE-COUNTER by a Procedure Division statement after an INITIATE statement has been executed.

6. PAGE-COUNTER is automatically incremented by 1 (one) each time a page break is recognized by the Report Writer, after the production of any PAGE or OVERFLOW FOOTING report group but before production of any PAGE or OVERFLOW HEADING report group.

6.11.6 LINE-COUNTER RULES

1. A LINE-COUNTER is a counter used by the Report Writer to determine when a PAGE/OVERFLOW HEADING and/or a PAGE/OVERFLOW FOOTING report group is to be presented. If a PAGE LIMIT(S) clause is written in the Report Description entry, a LINE-COUNTER is supplied for that report. The size of the numeric LINE-COUNTER is based on the number of lines specified per page in the PAGE LIMIT(S) clause.

2. If more than one Report Description entry exists in the Report Section, the user must qualify LINE-COUNTER by the report-name. LINE-COUNTER may be referred to in Data Division clauses and by Procedure Division statements.

3. Changing the LINE-COUNTER by Procedure Division statements may cause page format control to become unpredictable in the Report Writer.

4. LINE-COUNTER is automatically tested and incremented by the Report Writer based on control specifications in the PAGE LIMIT(S) clause and values specified in the LINE NUMBER and NEXT GROUP clauses.

5. LINE-COUNTER is automatically set to zero initially by the Report Writer; likewise, LINE-COUNTER is automatically reset to zero when PAGE LIMIT integer-1 LINES entry is exceeded during execution.

6. If a relative LINE NUMBER indication or relative NEXT GROUP indication exceeds the LAST DETAIL PAGE LIMIT specification during object time, that is, a page break, LINE-COUNTER is reset to zero. No additional setting based on the relative LINE NUMBER indication or NEXT GROUP indication that forced the page break takes place.

7. If an absolute LINE NUMBER indication or an absolute NEXT GROUP indication is equal to, or less than, the contents of the LINE-COUNTER during object time, the LINE-COUNTER is set to the absolute LINE NUMBER indication or the absolute NEXT GROUP indication following the implicit generation of any specified report groups.

8. The value of the LINE-COUNTER during any Procedure Division test statement represents the number of the last line used by the printing generated by the previous report group or, represents the number of the last line skipped by a previous NEXT GROUP specification.

9. The Report Writer LINE-COUNTER control prohibits the printing of successive report lines or report groups on the same line of the same page.

6.12 THE DATA DESCRIPTION--COMPLETE ENTRY SKELETON

6.12.1 FUNCTION

A data description entry specifies the characteristics of a particular item of data.

6.12.2 GENERAL FORMAT

Format 1

```
01 data-name-1; COPY library-name

┌             ┌word-1      ┐    ┌word-2      ┐
│ REPLACING   ┤            ├ BY ┤            ├
└             └identifier-1┘    └identifier-2┘

          ┌ ┌word-3      ┐    ┌word-4      ┐ ┐     ┐
          │,┤            ├ BY ┤            ├ │ ... │
          └ └identifier-3┘    └identifier-4┘ ┘     ┘
```

Format 2

```
level-number {data-name-1}
             {FILLER     }

[; REDEFINES data-name-2]

┌ ; {PICTURE} IS character-string [ DEPENDING ON data-name-3] ┐
└   {PIC    }                                                  ┘

┌                  ┌ COMPUTATIONAL   ┐ ┐
│                  │ COMP            │ │
│                  │ COMPUTATIONAL-n │ │
│ ; [ USAGE IS ]   ┤ COMP-n          ├ │
│                  │ DISPLAY         │ │
│                  │ DISPLAY-n       │ │
│                  │ INDEX           │ │
└                  └ INDEX-n         ┘ ┘
```

DATA DESCRIPTION SKELETON

Format 2 (Contd.)

```
┌ ; OCCURS integer-1 TIMES                                                        ┐
│                                                                                 │
│        ┌ { ASCENDING  }                                                ┐        │
│        │ {            } KEY IS data-name-4  [, data-name-5] ...        │ ...    │
│        └ { DESCENDING }                                                ┘        │
│                                                                                 │
│        [ INDEXED BY index-name-1  [, index-name-2] ... ]                        │
│                                                                                 │
│ ; OCCURS integer-2 TO integer-3 TIMES [DEPENDING ON data-name-6]                │
│                                                                                 │
│        ┌ { ASCENDING  }                                                ┐        │
│        │ {            } KEY IS data-name-4  [, data-name-5] ...        │ ...    │
│        └ { DESCENDING }                                                ┘        │
│                                                                                 │
└        [ INDEXED BY index-name-1  [, index-name-2] ... ]                        ┘

┌ ; { SYNCHRONIZED } [ LEFT  ] ┐
└   { SYNC         } [ RIGHT ] ┘

┌ ; { JUSTIFIED } RIGHT ┐
└   { JUST      }       ┘

┌ ;  RANGE IS literal-1 { THRU    } literal-2 ┐
└                       { THROUGH }           ┘

[ ; BLANK WHEN ZERO ]

[ ; VALUE IS literal-3 ] .
```

Format 3

```
66 data-name-1; RENAMES data-name-2 ┌ { THRU    } data-name-3 ┐ .
                                    └ { THROUGH }             ┘
```

Format 4

```
88 condition-name ; { VALUE IS   } literal-1 ┌ { THRU    } literal-2 ┐
                    { VALUES ARE }           └ { THROUGH }           ┘

        ┌ , literal-3 ┌ { THRU    } literal-4 ┐ ┐ ...   .
        └             └ { THROUGH }           ┘ ┘
```

6.12.3 SYNTAX RULES

1. All semicolons and commas are optional in the data description (except as noted in 6.31.5, Editing Rule 5), but the entry must be terminated by a period.

2. Format 4 cannot be used in the Constant Section.

3. Level-number in Format 2 may be any number from 01-49 or 77.

4. The clauses may be written in any order with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

5. The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

6. The words THRU and THROUGH are equivalent.

6.12.4 GENERAL RULES

1. The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, RANGE and BLANK WHEN ZERO must not be specified except at the elementary item level.

2. Format 4 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 4 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular condition-variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:

   a. Another condition-name.
   b. A level 66 item.
   c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).
   d. An index data item.

3. Format 1 is used when the COBOL library contains the 01 Record Description entry.

4. For a discussion of the COPY function, see Chapter 9, The COBOL Library.

6.13 THE REPORT GROUP DESCRIPTION-- COMPLETE ENTRY SKELETON

6.13.1 FUNCTION

The Report Group Description entry specifies the characteristics of a particular report group and of the individual data-name items within a report group.

6.13.2 GENERAL FORMAT

Format 1

```
01  [data-name]; COPY library-name

[ REPLACING { word-1       } BY { word-2       }
            { identifier-1 }    { identifier-2 }

        [ , { word-3       } BY { word-4       } ] ... ] .
            { identifier-3 }    { identifier-4 }
```

Format 2

```
01  [data-name-1]

[ ; LINE NUMBER IS { integer-1           } ]
                   { PLUS integer-2      }
                   { NEXT PAGE           }

[ ; NEXT GROUP IS  { integer-3           } ]
                   { PLUS integer-4      }
                   { NEXT PAGE           }
```

Format 2 (Cont.)

$$; \underline{\text{TYPE}}\ \text{IS} \left\{ \begin{array}{l} \underline{\text{REPORT}}\ \underline{\text{HEADING}} \\ \underline{\text{RH}} \\ \underline{\text{PAGE}}\ \underline{\text{HEADING}} \\ \underline{\text{PH}} \\ \underline{\text{OVERFLOW}}\ \underline{\text{HEADING}} \\ \underline{\text{OH}} \\ \left\{ \begin{array}{l} \underline{\text{CONTROL}}\ \underline{\text{HEADING}} \\ \underline{\text{CH}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-1} \\ \underline{\text{FINAL}} \end{array} \right\} \\ \underline{\text{DETAIL}} \\ \underline{\text{DE}} \\ \left\{ \begin{array}{l} \underline{\text{CONTROL}}\ \underline{\text{FOOTING}} \\ \underline{\text{CF}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-2} \\ \underline{\text{FINAL}} \end{array} \right\} \\ \underline{\text{OVERFLOW}}\ \underline{\text{FOOTING}} \\ \underline{\text{OV}} \\ \underline{\text{PAGE}}\ \underline{\text{FOOTING}} \\ \underline{\text{PF}} \\ \underline{\text{REPORT}}\ \underline{\text{FOOTING}} \\ \underline{\text{RF}} \end{array} \right\}$$

$$\left[;\ [\underline{\text{USAGE}}\ \text{IS}] \left\{ \begin{array}{l} \underline{\text{DISPLAY}} \\ \underline{\text{DISPLAY-n}} \end{array} \right\} \right].$$

Format 3

level-number [data-name-1]

$[;\ \underline{\text{COLUMN}}\ \text{NUMBER IS integer-1}]$

$[;\ \underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}]$

$[;\ \underline{\text{GROUP}}\ \text{INDICATE}]$

$$\left[; \left\{ \begin{array}{l} \underline{\text{JUSTIFIED}} \\ \underline{\text{JUST}} \end{array} \right\} \text{RIGHT} \right]$$

$$\left[;\ \underline{\text{LINE}}\ \text{NUMBER IS} \left\{ \begin{array}{l} \text{integer-2} \\ \underline{\text{PLUS}}\ \text{integer-3} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{array} \right\} \right]$$

$$\left[; \left\{ \begin{array}{l} \underline{\text{PICTURE}} \\ \underline{\text{PIC}} \end{array} \right\} \text{IS character-string} \right]$$

$$\left[;\ \underline{\text{RESET}}\ \text{ON} \left\{ \begin{array}{l} \text{identifier-1} \\ \underline{\text{FINAL}} \end{array} \right\} \right]$$

REPORT GROUP SKELETON

Format 3 (Cont.)

```
  { ; SOURCE IS [SELECTED] identifier-2                                    }
  { ; SUM identifier-3 [ , identifier-4]... [UPON data-name-2]             }
  { ; VALUE IS literal-1                                                   }

    [ ; [USAGE IS] { DISPLAY   } ]
    [              { DISPLAY-n } ]
```

6.13.3 SYNTAX RULES

1. All semicolons are optional in the Report Group Description but the entry must be terminated by a period. Except for the data-name clause, which must immediately follow the level-number when present, the clauses may be written in any order.

2. In order for a report group to be referred to by a Procedure Division statement it must have a data-name.

3. If the COLUMN NUMBER clause is present in the data description of an elementary item, the data description must also contain the PICTURE clause, in addition to one of the clauses SOURCE, SUM or VALUE.

4. In format 3, level-number may be any number from 01-49.

6.13.4 GENERAL RULES

1. Format 1 is used when the COBOL library contains the Report Group Description entry.

2. For a discussion of the COPY function see Chapter 9, The COBOL Library.

3. Format 2 is used to indicate a report group; the report group extends from this entry to the next report group level 01 entry.

4. Format 3 is used to indicate an elementary item or group item within a report group. If a report group is an elementary entry, Format 3 may include the TYPE and NEXT GROUP clauses in order to specify the report group and elementary item in the same entry.

5. When LINE NUMBER is specified in Format 2, entries for the first report line within the report group are presented on the specified line. When LINE NUMBER is specified in Format 3, sequential entries with the same level-number in the report group are implicity presented on the same line. A LINE NUMBER at a subordinate level must not contradict a LINE NUMBER at a group level.

6. The NEXT GROUP clause, when specified, refers to the spacing, at object time, between the last line of this report group and the first line of the next report group.

7. Variable length items must not be defined in the Report Section.

6.14 THE AREA CONTAINS CLAUSE

6.14.1 FUNCTION

The AREA CONTAINS clause specifies the size of a physical Saved Area.

6.14.2 GENERAL FORMAT

```
AREA CONTAINS integer-1   { RECORDS    }
                          { CHARACTERS }
```

6.14.3 3 SYNTAX RULES

1. Integer-1 must be an unsigned non-zero integer.

6.14.4 GENERAL RULES

1. One Saved Area record is automatically associated at object time with each out-of-line processing cycle. No more than one processing cycle has access to a single Saved Area record at any one time. The specific Saved Area record associated with each out-of-line processing cycle is released for further storage assignment upon the completion of the asynchronous processing cycle.

2. The execution of a PROCESS statement makes the associated Saved Area record unavailable for further reference with the in-line procedures.

3. When the CHARACTERS option is used, integer-1 indicates the number of characters of saved area that are available for storage of a variable number of saved records that can be asynchronously processed by associated PROCESS statements.

4. When the RECORDS option is used, integer-1 indicates the fixed number of saved records that can be asynchronously processed by associated PROCESS statements.

6.15 THE BLANK WHEN ZERO CLAUSE

6.15.1 FUNCTION

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

6.15.2 GENERAL FORMAT

BLANK WHEN ZERO

6.15.3 SYNTAX RULES

1. The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited. (See 6.31, The PICTURE clause.)

2. This clause cannot be used for variable length items.

6.15.4 GENERAL RULES

1. When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

6.16 THE BLOCK CLAUSE

6.16.1 FUNCTION

The BLOCK clause specifies the size of a physical record.

6.16.2 GENERAL FORMAT

$$\underline{\text{BLOCK}}\ \text{CONTAINS}\ \left[\text{integer-1}\ \underline{\text{TO}}\right]\ \text{integer-2}\ \left\{\begin{matrix}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{matrix}\right\}$$

6.16.3 SYNTAX RULES

1. Integer-1 and integer-2 must be unsigned non-zero integers.

6.16.4 GENERAL RULES

1. This clause is required except when:
    a. A physical record contains one and only one complete logical record.
    b. The hardware device assigned to the file has one and only one physical record size.
    c. The hardware device assigned to the file has more than one physical record size but the implementor has designated one as standard. In this case, the absence of this clause denotes the standard physical record size.
2. For a mass storage file the size may be stated in terms of RECORDS, unless one of the following situations exists, in which case the CHARACTERS option should be used:
    a. Logical records extend across physical records.
    b. The physical record contains padding (area not contained in a logical record).

c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

3. When the CHARACTERS option is used, the physical record size is specified in terms of the number of characters in Standard Data Format contained within the physical record, regardless of the types of characters used to represent the items within the physical record.

4. If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

5. If logical records of differing size are grouped into one physical record, the technique for determining the size of each logical record is specified by the implementor.

6.17 THE CODE CLAUSE

6.17.1 FUNCTION

The CODE clause defines a unique character or characters, which are to be affixed to each line of this report.

6.17.2 GENERAL FORMAT

CODE mnemonic-name-1

6.17.3 GENERAL RULES

1. CODE mnemonic-name-1 indicates a unique character or characters which is automatically affixed to and identifies each line of the report. More than one report may then be produced simultaneously onto one output device for later individual report selection. The implementor specifies how the unique character or characters is affixed to the report line.

6.18 THE COLUMN NUMBER CLAUSE

6.18.1 FUNCTION

The COLUMN NUMBER clause indicates the absolute column number on the printed page of the high-order (left-most) character of the elementary item, e.g., first print position of the item on the line.

6.18.2 GENERAL FORMAT

COLUMN NUMBER IS integer-1

6.18.3 SYNTAX RULES

1. Integer-1 must be an unsigned non-zero integer. (The first position of the print line is considered to be COLUMN NUMBER 1.)

2. The COLUMN NUMBER clause can only be given at the elementary level within a report group.

3. Within a report group and a particular LINE NUMBER specification, COLUMN NUMBER entries must be indicated from left to right.

6.18.4 GENERAL RULES

1. COLUMN NUMBER clause indicates that this elementary item is presented in the output report group; if COLUMN NUMBER is not indicated, the elementary item, though included in the description of the report group for control purposes, is suppressed when the report group is produced at object time.

6.19 THE CONTROL CLAUSE

6.19.1 FUNCTION

The CONTROL clause indicates the data-names which specify the control hierarchy for this report, that is, the control breaks.

6.19.2 GENERAL FORMAT

```
{CONTROL IS   }  { FINAL                                          }
{             }  { identifier-1  [, identifier-2] ...             }
{CONTROLS ARE }  { FINAL, identifier-1  [, identifier-2] ...      }
```

6.19.3 SYNTAX RULES

1. The identifiers specify the control hierarchy for this report and are listed in order from major to minor; FINAL is the highest control, identifier-1 is the major control, identifier-2 is the intermediate control, etc. The last identifier specified is the minor control.

2. The identifiers must be defined as an elementary item in the File, Working-Storage or Linkage Section of the Data Division.

6.19.4 GENERAL RULES

1. The CONTROL clause is required when CONTROL HEADING or CONTROL FOOTING report groups are specified.

2. The identifiers specified in the CONTROL clause are the only identifiers referred to by the RESET and TYPE clauses in a Report Group Description entry for this report. No identifier may be referred to by more than one TYPE CONTROL HEADING report group and one TYPE CONTROL FOOTING report group.

6.20 THE DATA-NAME OR FILLER CLAUSE

6.20.1 FUNCTION

A data-name specifies the name of the data being described. The word FILLER specifies an elementary item of the logical record that cannot be referred to directly.

6.20.2 GENERAL FORMAT

$$\left\{\begin{array}{l}\text{data-name}\\ \\ \underline{\text{FILLER}}\end{array}\right\}$$

6.20.3 SYNTAX RULES

1. In the File, Working-Storage, Constant and Linkage Sections, a data-name or the key word FILLER must be the first word following the level-number in each data description entry.

2. In the Report Section a data-name need not appear in a data description entry and FILLER must not be used.

6.20.4 GENERAL RULES

1. The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to directly.

2. In the Report Section, data-name must be given in the following cases:

   a. When the data-name represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division.

b. When reference is to be made to the SUM counter in the Procedure Division or Report Section.

c. When the SELECTED option is included with the SOURCE clause at a higher level to indicate at this lower level the SOURCE data-names which are to be used as elementary items.

6.21 THE DATA RECORDS CLAUSE

6.21.1 FUNCTION

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

6.21.2 GENERAL FORMAT

$$\underline{\text{DATA}} \quad \left\{ \begin{array}{l} \underline{\text{RECORD}}\text{ IS} \\ \underline{\text{RECORDS}}\text{ ARE} \end{array} \right\} \quad \text{data-name-1 [, data-name-2] ...}$$

6.21.3 SYNTAX RULES

1. Data-name-1 and data-name-2 are the names of data records and must have 01 level-number record descriptions, with the same names, associated with them.

6.21.4 GENERAL RULES

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

6.22 THE GROUP INDICATE CLAUSE

6.22.1 FUNCTION

The GROUP INDICATE clause indicates that this elementary item is to be produced only on the first occurrence of the item after any CONTROL or PAGE break.

6.22.2 GENERAL FORMAT

GROUP INDICATE

6.22.3 GENERAL RULES

1. The GROUP INDICATE clause must only be given at the elementary item level within a TYPE DETAIL report group.
2. An elementary item is not only group indicated in the first DETAIL report group containing the item after a control break, but is also group indicated in the first DETAIL report group containing the item on a new page, even though a control break did not occur.

6.23 THE JUSTIFIED CLAUSE

6.23.1 FUNCTION

The JUSTIFIED clause specifies non-standard positioning of data within a receiving data item.

6.23.2 GENERAL FORMAT

$$\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\} \quad \text{RIGHT}$$

6.23.3 SYNTAX RULES

1. The JUSTIFIED clause can be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.

6.23.4 GENERAL RULES

1. The standard rules for positioning data within an elementary item are:
   a. If the receiving data item is described as numeric;
      (1) The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.
      (2) When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its right-most character and is aligned as in (1) above.
   b. If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeroes.

c. If the receiving data item is alphanumeric (other than a numeric edited data item) or alphabetic, the sending data is moved to the receiving character positions and aligned at the left-most character position in the data item with space fill or truncation to the right.

2. The JUSTIFIED clause cannot be specified for a numeric edited data item or for an item described as numeric.

3. The JUSTIFIED clause cannot be specified for an item whose size is variable.

4. When the receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the left-most characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the right-most character position in the data item with space fill.

6.24 THE LABEL RECORDS CLAUSE

6.24.1 FUNCTION

The LABEL RECORDS clause specifies whether labels are present and, if present, identifies the label.

6.24.2 GENERAL FORMAT

$$\underline{\text{LABEL}} \quad \left\{ \begin{array}{l} \underline{\text{RECORDS}}\ \text{ARE} \\ \underline{\text{RECORD}}\ \text{IS} \end{array} \right\} \quad \left\{ \begin{array}{l} \underline{\text{OMITTED}} \\ \underline{\text{STANDARD}} \\ \text{data-name-1} \quad [\text{, data-name-2}] \ldots \end{array} \right\}$$

6.24.3 GENERAL RULES

1. This clause is required in every File Description entry.

2. OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

3. STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the implementor's label specifications.

4. Data-name-1, data-name-2, etc., are names of label records and must not appear in the DATA RECORDS clause and must be the subject of a Record Description associated with the file.

5. All Procedure Division references to the data-names specified in this clause, or to any items subordinate to these data-names, must appear within USE Procedures.

6.25 LEVEL-NUMBER

6.25.1 FUNCTION

The level-number shows the hierarchy of data within a logical record or report group. In addition, it is used to identify entries for condition-names, noncontiguous Constant, Working-Storage and Linkage items and the RENAMES clause.

6.25.2 GENERAL FORMAT

level-number

6.25.3 SYNTAX RULES

1. A level-number is required as the first element in each data description entry.

2. Data description entries subordinate to an FD, SD or SA entry may have level-numbers with the values 01 through 49, 66 and 88.

3. Data description entries subordinate to an RD entry may have level-numbers with the values 01 thru 49 only.

4. Multiple level 01 entries subordinate to a particular level indicator in the File Section represent implicit redefinitions of the same area.

5. An entry with a level-number of 88 cannot be used in the Constant Section.

6.25.4 GENERAL RULES

1. The level-number 01 identifies the first entry in each Record Description or a report group.

2. Special level-numbers have been assigned to certain entries where there is no real concept of level:

LEVEL-NUMBER

a. Level-number 66 is assigned to identify RENAMES entries and can be used only as described by Format 3 of the Data Description Skeleton (see 6.12.2, Data Description Skeleton).

b. Level-number 77 is assigned to identify noncontiguous constants and noncontiguous Working-Storage and Linkage data items and can be used only as described by Format 2 of the Data Description Skeleton (see 6.12.2, Data Description Skeleton).

c. Level-number 88 is assigned to entries which define condition-names associated with a conditional variable and can be used only as described by Format 4 of the Data Description Skeleton (see 6.12.2, Data Description Skeleton).

6.26 THE LINAGE CLAUSE

6.26.1 FUNCTION

The LINAGE clause specifies the number of lines to be written on a logical printer page.

6.26.2 GENERAL FORMAT

$$\underline{\text{LINAGE}}\ \text{IS} \left\{ \begin{array}{l} \text{identifier-1 } \underline{\text{LINES}} \\ \text{integer-1 } \underline{\text{LINES}} \\ \text{mnemonic-name} \end{array} \right\}$$

6.26.3 SYNTAX RULES

1. When identifier-1 or integer-1 is used, the data description must be that of a numeric elementary item without any positions to the right of the assumed decimal point.

2. When the mnemonic-name option is used, the name is identified with an end-of-page feature specified by the implementor. The mnemonic-name is defined in the SPECIAL-NAMES paragraph in the Environment Division.

3. The LINAGE clause and the REPORT clause are not allowed in the same FD entry.

6.26.4 GENERAL RULES

1. The LINAGE clause provides a means of specifying the depth of a printed page; the printed page may or may not be equal to the physical perforated continuous form often associated with the page length.

2. The value of integer-1, as specified in the LINAGE clause, will be used, at the time the file is opened by the OPEN statement, to specify the number of lines (written and/or spaced) on a printed page.

3. The value of identifier-1, as specified in the LINAGE clause, will be used, at each end-of-page, to specify the number of lines (written and/or spaced) for the next printed page.

4. A LINAGE-COUNTER is generated by the presence of the LINAGE clause. The rules governing the LINAGE-COUNTER are as follows:

a. One LINAGE-COUNTER is supplied for each file described in the File Section whose FD entry includes the LINAGE clause.

b. LINAGE-COUNTER may be referred to by Procedure Division statements. Since more than one LINAGE-COUNTER may exist, the user must qualify LINAGE-COUNTER by the file-name when necessary.

c. The user is responsible for page format control if the value of the LINAGE-COUNTER is changed by Procedure Division statements.

d. LINAGE-COUNTER is automatically incremented each time a WRITE statement is executed for the associated file, unless the mnemonic-name option of the WRITE ADVANCING statement is used.

  (1) When the ADVANCING phrase of the WRITE statement is used with the identifier-2 LINES option, the increment is the value of identifier-2.

  (2) When the ADVANCING phrase of the WRITE statement is used with integer LINES option, the increment is the value of integer.

  (3) When the ADVANCING phrase of the WRITE statement is absent, the increment value is one (1). (See 7.42.4.4, The WRITE Statement.)

e. LINAGE-COUNTER is automatically set to zero initially by the OPEN statement and likewise is automatically reset to zero when the value in the LINAGE clause entry is exceeded during execution.

f. The value of the LINAGE-COUNTER at any given time represents the last line number printed or spaced on a logical printer page.

g. If the mnemonic-name option of the WRITE ... ADVANCING statement is used the contents of the LINAGE-COUNTER may be unpredictable.

6.27 THE LINE NUMBER CLAUSE

6.27.1 FUNCTION

The LINE NUMBER clause indicates the absolute or relative line number of this entry in reference to the page or the previous entry.

6.27.2 GENERAL FORMAT

```
                   ⎧ integer-1         ⎫
LINE NUMBER   IS   ⎨ PLUS integer-2    ⎬
                   ⎩ NEXT PAGE         ⎭
```

6.27.3 SYNTAX RULES

1. Integer-1 and integer-2 must be unsigned non-zero integers. Integer-1 must be within the range specified by the PAGE LIMITS clause in the Report Description entry.

2. The LINE NUMBER clause must be given for each report line of a report group. For the first line of a report group it must be given either at the report group level or prior to or for the first elementary item in the line. For report lines other than the first in a report group, it must be given prior to or for the first elementary item in the line.

6.27.4 GENERAL RULES

1. If the LINE NUMBER clause is specified at the report group level, entries for the first report line within the report group are presented on the specified line number. If LINE NUMBER is specified for an entry on other than the report group level, sequential entries following that entry within the report group with the same level-number are implicitly presented on the same line number. A LINE NUMBER at a subordinate level may not contradict a LINE NUMBER at a group level.

2. Within a report group, absolute LINE NUMBER entries must be indicated in ascending order, and an absolute LINE NUMBER cannot be preceded by a relative LINE NUMBER.

3. The NEXT PAGE phrase may be used to indicate an automatic skip to the next page before presenting the first line of the current report group. Appropriate TYPE PAGE/OVERFLOW FOOTINGS and TYPE PAGE/OVERFLOW HEADINGS will be produced as specified.

4. Integer-1 indicates an absolute line number which sets the LINE-COUNTER to this value for printing the item in this entry, and following entries within the report group, until a different value for the LINE-COUNTER is specified.

5. Integer-2 indicates a relative line number which increments the LINE-COUNTER for printing the item in this entry, and following entries within the report group, until a different value for the LINE-COUNTER is specified.

6.28 THE NEXT GROUP CLAUSE

6.28.1 FUNCTION

The NEXT GROUP clause indicates the spacing condition following the last line of the report group.

6.28.2 GENERAL FORMAT

$$\underline{\text{NEXT}}\ \underline{\text{GROUP}}\ \text{IS} \left\{\begin{array}{l} \text{integer-1} \\ \underline{\text{PLUS}}\ \text{integer-2} \\ \underline{\text{NEXT}}\ \underline{\text{PAGE}} \end{array}\right\}$$

6.28.3 SYNTAX RULES

1. Integer-1 and integer-2 must be unsigned non-zero integers. Integer-1 cannot exceed the maximum number of lines specified per report page.

6.28.4 GENERAL RULES

1. Integer-1 indicates an absolute line number which sets the LINE-COUNTER to this value after producing the last line of the current report group.

2. Integer-2 indicates a relative line number which increments the LINE-COUNTER by the integer-2 value. Integer-2 represents the number of lines skipped following the last line of the current report group. Further spacing is specified by the LINE NUMBER clause of the next report group produced.

3. The NEXT PAGE clause indicates an automatic skip to the next page following the generation of the last line of the current report group. Appropriate page/overflow footings and page/overflow headings will be produced as specified.

4. The NEXT GROUP clause must appear only at the 01 level which defines the report group. When specified for a CONTROL FOOTING/HEADING report group, the NEXT GROUP clause results in automatic line spacing only when a control break occurs on the level for which that control is specified.

6.29 THE OCCURS CLAUSE

6.29.1 FUNCTION

The OCCURS clause eliminates the need for separate entries for repeated data and supplies information required for the application of subscripts or indices.

6.29.2 GENERAL FORMAT

Format 1

```
OCCURS integer-2 TIMES [ { ASCENDING  }   KEY IS data-name-2
                         { DESCENDING }
 [, data-name-3 ] ...]   ... [INDEXED BY index-name-1
        [ , index-name-2 ] ...]
```

Format 2

```
OCCURS integer-1 TO integer-2 TIMES  [DEPENDING ON data-name-1]
 [ { ASCENDING  } KEY IS data-name-2  [ , data-name-3 ]  ... ] ...
   { DESCENDING }
 [INDEXED BY index-name-1  [ , index-name-2 ]  ...]
```

6.29.3 SYNTAX RULES

1. Integer-1 and integer-2 must be positive integers. Where both are used the value of integer-1 must be less than integer-2. The value of integer-1 may be zero, but integer-2 cannot be zero.

2. The data description of data-name-1 must describe a positive integer.

3. Data-name-1, data-name-2, data-name-3, ... may be qualified.

4. Data-name-2 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.

5. Data-name-3, etc. must be the name of an entry subordinate to the group item which is the subject of this entry.

6. An INDEXED BY phrase is required if the subject of this entry, or an item within it if it is a group item, is to be referred to by indexing. The index-name identified by this clause is not defined elsewhere since its allocation and format are dependent on the hardware and, not being data, cannot be associated with any data hierarchy.

7. The DEPENDING option is only required when the end of the occurrences cannot otherwise be determined.

6.29.4 GENERAL RULES

1. The OCCURS clause cannot be specified in a data description entry that:

   a. Has a 01 or a 77 level-number.

   b. Describes an item whose size is variable. The size of an item is variable if its data description, or if any item subordinate to it, contains an 'L' in the PICTURE clause, or if the data description of any subordinate item contains Format 2 of the OCCURS clause.

2. The OCCURS clause is used in defining tables and other homogeneous sets of repeated data. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than SEARCH. Further, if the subject of this entry is the name of a group item, then all data-names belonging to the group must be subscripted or indexed whenever they are used as operands.

3. The data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

4. In Format 1, the value of integer-2 represents the exact number of occurrences. In Format 2, the value of integer-2 represents the maximum number of occurrences.

5. Format 2 specifies that the subject of this entry has a variable number of occurrences. Integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject is variable, but that the number of occurrences is variable.

6. The value of data-name-1 is the count of the number of occurrences of the subject and its value must not exceed integer-2. Reducing the value of data-name-1 makes the contents of data items, whose occurrence numbers now exceed the value of data-name-1, unpredictable.

7. If data-name-1 is an entry in the same record as the current data description entry, the data description entry for data-name-1 must be prior to a data description entry containing the OCCURS clause in which data-name-1 appears.

8. Each implementor will specify whether unused character positions resulting from the DEPENDING option will appear in the external media.

9. Any entry which contains or has a subordinate entry which contains Format 2 cannot be the object of the REDEFINES clause.

10. The KEY IS option is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. The data-names are listed in their descending order of significance.

11. If data-name-2 is not the subject of this entry, then:

    a. All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of this entry.

    b. None of the items identified by data-names in the KEY IS phrase can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

6.30 THE PAGE LIMIT CLAUSE

6.30.1 FUNCTION

The PAGE LIMIT clause indicates the specific line control to be maintained within the logical presentation of a page.

6.30.2 GENERAL FORMAT

$$\underline{\text{PAGE}} \quad \left\{ \begin{matrix} \text{LIMIT IS} \\ \text{LIMITS ARE} \end{matrix} \right\} \quad \text{integer-1} \quad \left\{ \begin{matrix} \underline{\text{LINE}} \\ \underline{\text{LINES}} \end{matrix} \right\}$$

$$[\text{ , } \underline{\text{HEADING}} \text{ integer-2 }] \qquad [\text{ , } \underline{\text{FIRST}} \ \underline{\text{DETAIL}} \text{ integer-3 }]$$

$$[\text{ , } \underline{\text{LAST}} \ \underline{\text{DETAIL}} \text{ integer-4 }] \qquad [\text{ , } \underline{\text{FOOTING}} \text{ integer-5 }]$$

6.30.3 SYNTAX RULES

1. Integer-1 through integer-5 must be unsigned non-zero integers.
2. Integer-2 through integer-5 each must either be less than or equal to integer-1.

6.30.4 GENERAL RULES

1. The PAGE LIMIT clause is required when page format must be controlled by the Report Writer. The PAGE LIMIT clause may be omitted when no association is desired between report groups and the physical format of an output page.

2. The PAGE LIMIT integer-1 LINES clause is required to specify the depth of the report page; the depth of the report page may or may not be equal to the physical perforated continuous for often associated in a report with the page length.

PAGE LIMIT

3. LINE-COUNTER must be able to contain the value specified by integer-1.

4. If absolute line spacing is indicated for all the report group(s), none of the integer-2 through integer-5 controls need to be specified.

5. If relative spacing is indicated for individual TYPE DETAIL Report Group entries, some or all of the above limits must be defined, dependent on the type of report groups within the report, in order for the Report Writer to maintain control of page format.

HEADING integer-2: the first line number of the first heading print group. No print group will start preceding integer-2.

FIRST DETAIL integer-3: the first line number of the first normal print group, that is, body; no DETAIL or CONTROL print group will start before integer-3.

LAST DETAIL integer-4: the last line number of the last normal print group, that is, body; no DETAIL or CONTROL HEADING print group will extend beyond integer-4.

FOOTING integer-5: the last line number of the last CONTROL FOOTING print group; no CONTROL FOOTING print group will start before integer-3 nor extend beyond integer-5. TYPE PAGE FOOTING or TYPE OVERFLOW FOOTING print groups will follow integer-5.

6. When relative line numbers are specified for report groups, PAGE LIMITS integer-1 is specified and some or all of the HEADING integer-2, FIRST DETAIL integer-3, LAST DETAIL integer-4, FOOTING integer-5 clauses are omitted, the following implicit control is assumed for the omitted specifications:

   a. If HEADING integer-2 is omitted, integer-2 is considered to be equivalent to the value one (1), that is, LINE NUMBER one.

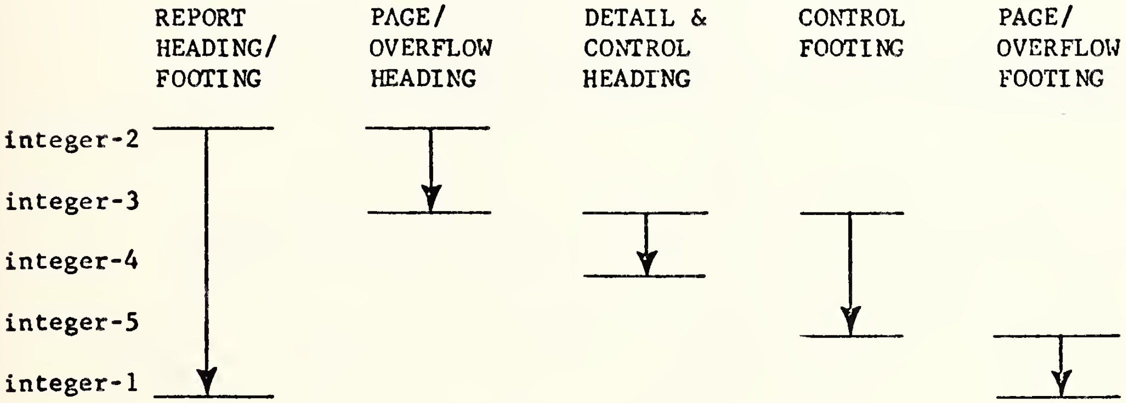
   b. If FIRST DETAIL integer-3 is omitted, integer-3 is considered to be equivalent to the value of integer-2.

c. If LAST DETAIL integer-4 is omitted, integer-4 is considered to be equivalent to the value of integer-5.

d. If FOOTING integer-5 is cmitted, integer-5 is considered to be equivalent to the value of integer-4.

e. If both LAST DETAIL integer-4 and FOOTING integer-5 are omitted, integer-4 and integer-5 both are considered to be equivalent to the value of integer-1.

7. The following chart pictorially represents page format report group control when the PAGE LIMIT clause is specified:



8. Absolute LINE NUMBER or absolute NEXT GROUP spacing, (see Report Group Description Entry), must be consistent with controls specified in the PAGE LIMIT clause.

9. Only one PAGE LIMIT clause may be specified per Report Description entry.

6.31 THE PICTURE CLAUSE

6.31.1 FUNCTION

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

6.31.2 GENERAL FORMAT

$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}$ IS character-string $\left[\underline{\text{DEPENDING}}\text{ ON data-name}\right]$

6.31.3 SYNTAX RULES

1. A PICTURE clause can be specified only at the elementary item level.

2. A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.

3. The maximum number of symbols allowed in the character-string is 30.

4. The DEPENDING ON clause is used to denote a variable length elementary item. Variable length items cannot be described in the REPORT SECTION.

5. The data description of data-name must be such that it defines a positive integer. The value of data-name represents the number of characters in the item being described and may have a value of zero. If data-name appears in a record of a file, its least significant character position must always be the same number of character positions from the beginning of the record.

6. The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

7. Data-name may be qualified.

8. PIC is an abbreviation for PICTURE.

9. The asterisk when used as the Zero Suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.

6.31.4 General Rules

1. There are five categories of data that can be described with a PICTURE clause; ALPHABETIC, NUMERIC, ALPHANUMERIC, Alphanumeric Edited and Numeric Edited.

2. To define an item as ALPHABETIC:

   a. Its PICTURE character-string can only contain the symbols 'A', 'B', 'J', 'K', 'L'; and

   b. Its contents when represented in Standard Data Format must be any combination of the twenty-six (26) letters of the Roman alphabet and the space from the COBOL character set.

3. To define an item as NUMERIC:

   a. Its PICTURE character-string can only contain the symbols '0', '9', 'J', 'K', 'L', 'P', 'S', and 'V'; and

   b. Its contents when represented in Standard Data Format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9', and the item may include an operational sign.

4. To define an item as ALPHANUMERIC:

   a. Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'J', 'K', 'L', 'X', '9', and the item is treated as if the character-string contained all 'X's. A PICTURE character-string which contains all 'A's or all '9's, with or without the symbols 'J', 'K', or 'L', does not define an ALPHANUMERIC item, and;

   b. Its contents when represented in Standard Data Format are allowable characters in the computer's character set.

5. To define an item as Alphanumeric Edited:

   a. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'J', 'K', 'X', '9', 'B', and '0'; and

      (1) The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X', or;

      (2) The character-string must contain at least one '0' (zero) and at least one 'A'; and

   b. Its contents when represented in Standard Data Format are allowable characters in the computer's character set.

6. To define an item as Numeric Edited:

   a. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', 'J', 'K', 'P', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules. The maximum number of digit positions that may be represented in the character-string is 18; and

   b. The contents of the character positions of these symbols that are allowed to represent a digit in Standard Data Format, must be one of the numerals.

7. The size of an elementary item where size means the number of character positions occupied by the elementary item in Standard Data Format, is determined by the number of allowable symbols that represent character positions. An unsigned non-zero integer which is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'P', 'Z', '*', 'B', '0', '+', '-', or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE, 'S', 'V', 'L', '.', 'CR', and 'DB'. The number of occurrences within a PICTURE of the symbols 'J' and 'K' is determined by the implementor.

8. The function of the symbols used to describe an elementary item are explained as follows:

A Each 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

B Each 'B' in the character-string represents a character position into which the space character will be inserted.

J,K Indicates data control symbols the functions of which are specified by the individual implementor.

L The 'L' must appear as the left-most character in the character-string of every elementary item whose length is variable. If the implementor's standard method of determining the end of variable length items is used, the DEPENDING ON clause is not necessary. The PICTURE designates the maximum size of the item. The 'L' is not counted in determining the size of the item.

P The 'P' indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items which appear as operands in arithmetic statements. The scaling position character 'P' can appear only to the left or right as a continuous string of 'P's within a PICTURE description; since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are left-most PICTURE characters and to the right of 'P's if 'P's are right-most PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the left-most or right-most character within such a PICTURE description.

S The letter 'S' is used in a character-string to indicate the presence of an operational sign and must be written as the left-most character in the PICTURE, exclusive of the symbol 'L'. The 'S' is not counted in determining the size of the elementary item.

V The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the right-most symbol in the string the 'V' is redundant.

X Each 'X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set.

Z Each 'Z' in a character-string may only be used to represent the left-most leading numeric character positions which will be replaced by a space character when the contents of that character position is zero. Each 'Z' is counted in the size of the item.

9 Each '9' in the character-string represents a character position which contains a numeral and is counted in the size of the item.

0 Each '0' (zero) in the character-string represents a character position into which the numeral zero will be inserted. The '0' is counted in the size of the item.

, Each ',' (comma) in the character-string represents a character position into which the character ',' will be inserted. This character position is counted in the size of the item.

. When the character '.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

+,-,CR,DB These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data-item.

* Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position is zero. Each '*' is counted in the size of the item.

cs The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the currency sign or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

6.31.5 EDITING RULES

1. There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available. They are:

    a. Simple insertion

    b. Special insertion

    c. Fixed insertion

    d. Floating insertion

    There are two types of suppression and replacement editing:

    a. Zero suppression and replacement with spaces

    b. Zero suppression and replacement with asterisks

2. The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given category:

Category	Type of Editing
ALPHABETIC	Simple Insertion, 'B' only
NUMERIC	Simple Insertion, '0' only
ALPHANUMERIC	None
Alphanumeric Edited	Simple Insertion, '0' and 'B'
Numeric Edited	All, subject to rules in rule 3, below.
Any Variable Length Item	None

3. Floating Insertion editing and editing by Zero Suppression and Replacement are mutually exclusive in a PICTURE clause. Only one type of Replacement may be used with Zero Suppression in a PICTURE clause.

4. Simple Insertion Editing.

   The ',' (comma), 'B' (space) and '0' (zero) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

5. Special Insertion Editing.

   The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. If the insertion character is the last symbol in the character string and additional clauses follow the character-string, then the character-string must be immediately followed by the semicolon punctuation character, followed by a space. If the PICTURE clause is the last clause of that Data Division entry, and the insertion character is the last symbol in the character-string, the insertion character must be immediately followed by a period punctuation character, followed by a space. This results in two consecutive periods appearing in the data description entry. The result of Special Insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

6. Fixed Insertion Editing.

   The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB' are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the right-most character positions that are counted in the size of the item. The symbol '+' or '-', when used, must be the left-most or right-most character position to be counted in the size of the item. The currency symbol must be the left-most character position to be counted in the size of the item except that it can be preceded by either a '+' or a '-'

symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string. Editing sign control symbols produce the following results depending upon the value of the data item:

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT DATA ITEM POSITIVE OR ZERO	 DATA ITEM NEGATIVE
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

7. Floating Insertion Editing.

The currency symbol and editing sign control symbols '+' or '-' are the insertion characters and they are mutually exclusive as floating insertion characters in a given PICTURE character-string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the allowable insertion characters to represent the left-most numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters embedded in the string of floating insertion characters or to the immediate right of this string are part of the floating string.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

The result of floating insertion editing depends upon the representation in the PICTURE character-string. If the insertion characters are only to the left of the decimal point the result is a single insertion character that will be placed into the character position immediately preceding the decimal point, or the first non-zero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of non-floating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

8. Zero Suppression Editing.

The suppression of leading zeroes in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used the replacement character will be the space and if the asterisk is used, the replacement character will be '*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero the entire data item will be spaces if the suppression symbol is 'Z' or all '*', except for the actual decimal point, if the suppression symbol is'*'.

9. The symbols '+', '-', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

   The following chart shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces { } indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol 'cs'.

PICTURE

		NON-FLOATING INSERTION SYMBOLS								OTHER SYMBOLS												
		B	0	,	.	{+ -}	{+ -}	{CR DB}	cs	A X	L	P	P	S	V	{Z *}	{Z *}	9	{+ -}	{+ -}	cs	cs
NON-FLOATING INSERTION SYMBOLS	B	x	x	x	x	x			x	x		x			x	x	x	x	x	x	x	x
	0	x	x	x	x	x			x	x		x		x	x	x	x	x	x	x	x	x
	,	x	x	x	x	x			x			x			x	x	x	x	x	x	x	x
	.	x	x	x		x			x			x				x		x	x		x	
	{+ -}											x										
	{+ -}	x	x	x	x				x			x			x	x	x	x			x	x
	{CR DB}	x	x	x	x				x			x			x	x	x	x			x	x
	cs					x						x			x							
OTHER SYMBOLS	A X	x	x							x	x							x				
	L																					
	P										x	x		x	x							
	P	x	x	x		x	x	x	x		x		x	x		x		x	x		x	
	S										x											
	V	x	x	x		x			x		x		x	x		x		x	x		x	
	{Z *}	x	x	x		x			x							x						
	{Z *}	x	x	x	x	x			x			x			x	x	x					
	9	x	x	x	x	x			x	x	x	x		x	x	x		x	x		x	
	{+ -}	x	x	x					x										x			
	{+ -}	x	x	x	x				x			x			x				.x	x		
	cs	x	x	x		x															x	
	cs	x	x	x	x	x						x			x						x	x

At least one of the symbols 'A', 'X', 'Z', '9' or '*', or at least two of the symbols '+', '-' or 'cs' must be present in a PICTURE string.

Non-floating insertion symbols '+' and '-', and other symbol 'P' appear twice. The left-most column and upper-most row represents their use to the left of the PICTURE's numeric character positions and the second their use to the right of the PICTURE's numeric character positions. Non-floating insertion symbols '+' and '-', and other symbols 'Z', '*', 'cs', '+', and '-', appear twice. The left-most column and upper-most row represents the use before the decimal position, the second the use after the decimal point position.

6.32 THE RANGE CLAUSE

6.32.1 FUNCTION

The RANGE clause indicates the potential range of the value of an item.

6.32.2 GENERAL FORMAT

$\underline{\text{RANGE}}$ IS literal-1 $\left\{\begin{array}{c}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}$ literal-2

6.32.3 SYNTAX RULES

1. The RANGE clause can be written only at the elementary item level.
2. Literal-1 and literal-2 may be figurative constants.
3. Literal-1 and literal-2 must not contain more digits than are specified in the PICTURE clause.
4. The words THRU and THROUGH are equivalent.

6.32.4 GENERAL RULES

1. This clause is used for documentation only.
2. For numeric items, literal-1 and literal-2 represent the respective minimum and maximum values of the item.
3. For nonnumeric items, each character of literal-1 and literal-2 represents the respective minimum and maximum values of the corresponding character position in the item.

6.33 THE RECORD CONTAINS CLAUSE

6.33.1 FUNCTION

The RECORD CONTAINS clause specifies the size of data records.

6.33.2 GENERAL FORMAT

```
RECORD CONTAINS  [ integer-1 TO ]  integer-2 CHARACTERS
```

6.33.3 SYNTAX RULES

1. Integer-1 and integer-2 must be unsigned non-zero integers.

6.33.4 GENERAL RULES

1. The size of each data record is completely defined within the Record Description entry, therefore this clause is never required. When present, however, the following notes apply:

    a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

    b. The size is specified in terms of the number of characters in Standard Data Format contained within the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in all variable length items subordinate to the record. This sum may be different from the actual size of the record; see 3.3.4, Selection of Character Representation and Radix; 6.40, The SYNCHRONIZED Clause; and 6.47, The USAGE Clause.

6.34 THE RECORDING MODE CLAUSE

6.34.1 FUNCTION

The RECORDING MODE clause specifies the format or organization of data on external media.

6.34.2 GENERAL FORMAT

RECORDING MODE IS mode-name

6.34.3 GENERAL RULES

1. The RECORDING MODE clause is necessary for computers having a data format or organization which may vary on external media.
2. When a computer has only one mode, this clause is not needed.
3. Each implementor will assign specific names to the alternative modes of data representation which can be handled. When a standard recording mode exists, the implementor may choose to assign names only to the nonstandard modes. In this case, the absence of the RECORDING MODE clause denotes the standard mode.

6.35 THE REDEFINES CLAUSE

6.35.1 FUNCTION

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

6.35.2 GENERAL FORMAT

level-number data-name-1 ; REDEFINES data-name-2

Level-number, data-name-1 and the semicolon are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the REDEFINES clause.

6.35.3 SYNTAX RULES

1. The REDEFINES clause, when specified, must immediately follow data-name-1.

2. The level-numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88.

3. This clause must not be used in level 01 entries in the File Section. Implicit redefinition is provided by the DATA RECORDS clause in the File Description entry.

6.35.4 GENERAL RULES

1. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

2. When the level-number of data-name-1 is other than 01, it must specify a storage area of the same size as data-name-2. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

3. Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must follow the entries defining the area being redefined, without intervening entries that define new storage areas. Multiple redefinitions of the same storage area must all use the data-name of the entry that originally defined the area.

4. The data description entry for data-name-2 cannot contain an OCCURS clause, nor can data-name-2 be subordinate to an entry which contains an OCCURS clause. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS clause (see 6.29.4.1.b, The OCCURS Clause).

5. The entries giving the new description of the storage area must not contain any VALUE clauses, except in condition-name entries.

6.36 THE RENAMES CLAUSE

6.36.1 FUNCTION

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

6.36.2 GENERAL FORMAT

$$66\ \text{data-name-1}\ ;\ \underline{\text{RENAMES}}\quad \text{data-name-2}\ \left[\begin{Bmatrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{Bmatrix}\quad \text{data-name-3}\right].$$

Level-number 66, data-name-1 and the semicolon are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the RENAMES clause.

6.36.3 SYNTAX RULES

1. All RENAMES entries associated with a given logical record must immediately follow its last data description entry.

2. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the associated logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 level entry.

3. Data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the level 01 or FD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

4. Data-name-2 must precede data-name-3 in the Record Description, and after any associated redefinition, the beginning point of the area described by data-name-3 must logically follow the beginning point of the area described by data-name-2.

5. Data-name-3 cannot be subordinate to data-name-2.

6. Data-name-2 and data-name-3 may be qualified.

7. The words THRU and THROUGH are equivalent.

6.36.4 GENERAL RULES

1. One or more RENAMES entries can be written for a logical record.

2. When data-name-3 is specified, data-name-1 is a group item which includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

3. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item; when data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

4. When data-name-3 is specified, none of the elementary items within the range, including data-name-2 and data-name-3, can be of variable length.

6.37 THE REPORT CLAUSE

6.37.1 FUNCTION

The REPORT clause cross references the Report Description entries with their associated File Description entry.

6.37.2 GENERAL FORMAT

$$\left\{\begin{array}{l}\underline{\text{REPORT}}\ \text{IS}\\ \underline{\text{REPORTS}}\ \text{ARE}\end{array}\right\}\quad \text{report-name-1}\quad [\ ,\ \text{report-name-2}\ ]\ \ldots$$

6.37.3 SYNTAX RULES

1. Each report-name listed in the FD entry must be the subject of a Report Description (RD) entry in the Report Section.

6.37.4 GENERAL RULES

1. The REPORT clause is required in the File Description entry if the file is an output report file or is to contain output report records.

2. The presence of more than one report-name indicates that the file contains more than one report. These reports may be of differing formats, but must be the same size. The order in which they are listed is not significant.

6.38 THE RESET CLAUSE

6.38.1 FUNCTION

The RESET clause indicates the CONTROL identifier that causes the SUM counter in the elementary item entry to be reset to zero on a CONTROL break.

6.38.2 GENERAL FORMAT

$$\underline{\text{RESET}}\ \text{ON}\ \left\{\begin{array}{l}\text{identifier-1}\\ \underline{\text{FINAL}}\end{array}\right\}$$

6.38.3 SYNTAX RULES

1. Identifier-1 must be one of the identifiers described in the CONTROL clause in the Report Description entry. Identifier-1 must be a higher level CONTROL identifier than the CONTROL identifier associated with the CONTROL FOOTING report group in which the SUM and RESET clauses appear.

2. The RESET clause may only be used in conjunction with a SUM clause at the elementary level.

6.38.4 GENERAL RULES

1. After presentation of the TYPE CONTROL FOOTING report group, the counters associated with the report group are reset automatically to zero unless an explicit RESET clause is given specifying reset based on a higher level control than the associated control for the report group.

2. The RESET clause may be used for progressive totaling of identifiers where subtotals of identifiers may be desired without automatic resetting upon producing the report group.

6.39 THE SOURCE, SUM, AND VALUE CLAUSES

6.39.1 FUNCTION

The SOURCE, SUM or VALUE clauses define the purpose of this report item within the report group.

6.39.2 GENERAL FORMAT

$$\left\{\begin{array}{l} \underline{\text{SOURCE}}\ \text{IS}\ [\ \underline{\text{SELECTED}}\ ]\ \ \text{identifier-1} \\ \underline{\text{SUM}}\ \text{identifier-2}\ \ [,\ \text{identifier-3}]\ \ \ldots\ [\ \underline{\text{UPON}}\ \text{data-name-1}\ ] \\ \underline{\text{VALUE}}\ \text{IS}\ \ \text{literal-1} \end{array}\right\}$$

6.39.3 SYNTAX RULES

1. Identifier-1, identifier-2, and identifier-3 must each indicate an item which appears in the File, Working-Storage, Constant or Linkage Section or is the name of a SUM counter in the Report Section.

2. SOURCE (without SELECTED), SUM and VALUE clauses can only be given at the elementary level. The SOURCE IS SELECTED clause can only be given at a group level.

3. When the SELECTED phrase is specified, identifier-1 represents a group item. The identifiers described at the elementary level in the source record then become SOURCE entries in the associated report group. The SELECTED elementary level identifiers must be unique data-names.

4. Literal-1 may be numeric, non-numeric, or a figurative constant.

5. Data-name-1 may be qualified.

6.39.4 GENERAL RULES

SOURCE

1. The SOURCE clause indicates a data item which is to be used as the source for this report item. This data item is called a SOURCE data item or a SOURCE item. The item is presented according to the PICTURE clause in the associated elementary report group entry.

2. The elementary level items within identifier-1 are matched against the data-names specified at the elementary level within the report group. Matching data-names are SELECTED as SOURCE item entries to be included and presented within the report group, according to the PICTURE and USAGE specifications given with the data-name in the report group entry.

SUM

3. A SUM clause may only appear in a TYPE CONTROL FOOTING report group.

4. If a SUM counter is referred to by a Procedure Division statement or Report Section entry, a data-name must be specified with the SUM clause entry. The data-name then represents the summation counter automatically generated by the Report Writer to total the operands specified immediately following the SUM. If a summation counter is never referred to, the counter need not be named explicitly by a data-name entry. A SUM counter is only algebraically incremented just before presentation of the TYPE DETAIL report group in which the item being summed appears as a SOURCE item.

5. Whether the SUM clause names the summation counter or not, the PICTURE clause must be specified for each SUM counter. Editing characters or the editing clause may be included in the description of a SUM counter. Editing of a SUM counter only occurs upon the presentation of that SUM counter. At all other times the SUM counter is treated as a numeric data-item. The SUM counter must be large enough to accommodate the summed quantity without truncation of integral digits.

6. Each item being summed, that is, identifier-2, identifier-3, etc., must appear as a SOURCE item in a TYPE DETAIL report group or be names of SUM counters in a TYPE CONTROL FOOTING report group at an equal or lower position in the control hierarchy. Although the items must be explicitly written in a TYPE DETAIL report group, they may be actually suppressed at presentation time. In this manner, direct association without ambiguity can be made from the current data available by a GENERATE statement to the data items to be presented within the Report Section.

7. If higher level report groups are indicated in the control hierarchy, counter updating, commonly called 'rolling counters forward', procedures take place prior to the reset operation.

8. The summation of data items defined as SUM counters in TYPE CONTROL FOOTING report groups is accomplished explicitly or implicitly with the Report Writer automatically handling the updating function. If a SUM control of a data item is not desired for presentation at a lower level but is desired for presentation at a higher level, the lower level SUM specification may be omitted. In this case, the same results are obtained as if the lower level SUM counter were specified.

9. The UPON data-name-1 phrase is required to obtain selective summation for a particular data item which is named as a SOURCE item in two or more TYPE DETAIL report groups. Identifier-2 and identifier-3 must be SOURCE data items in data-name-1. Data-name-1 must be the name of a TYPE DETAIL report group. If the UPON data-name-1 phrase is not used, identifier-2, identifier-3, etc., respectively, are added to the SUM counter at each execution of a GENERATE statement. This statement generates a TYPE DETAIL report group that contains the SUM operands at the elementary level.

   For further explanation, see 7.8, The ADD Statement.

6.40 THE SYNCHRONIZED CLAUSE

6.40.1 FUNCTION

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory (See 3.3.6, Item Alignment for Increased Object-Code Efficiency).

6.40.2 GENERAL FORMAT

$$\left\{ \begin{array}{l} \underline{\text{SYNCHRONIZED}} \\ \underline{\text{SYNC}} \end{array} \right\} \quad \left[\begin{array}{l} \underline{\text{LEFT}} \\ \underline{\text{RIGHT}} \end{array} \right]$$

6.40.3 SYNTAX RULES

1. This clause may only appear with an elementary item.

2. SYNC is an abbreviation for SYNCHRONIZED.

6.40.4 GENERAL RULES

1. This clause specifies that the COBOL processor, in creating the internal format of this item, must arrange the item in contiguous units of memory in such a way that no other data item appears in any of the memory units between the left and right natural boundaries delimiting this data item. If the size of the item is such that it does not, itself, utilize all of the memory between the delimiting natural boundaries, the unused memory units (or portions thereof) may not be used for any other data item. Such unused memory is, however, included in:

   a. The size of any group to which the elementary item belongs; and

   b. The computer storage area allocation when the elementary item appears as the object of a REDEFINES clause.

2. SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be positioned between natural boundaries in such a way as to effect efficient utilization of the elementary data item. The specific positioning is, however, determined by the implementor.

3. SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left boundary of the contiguous memory in which the elementary item is placed.

4. SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right boundary of the contiguous memory in which the elementary item is placed.

5. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining any action that depends on size, such as justification, truncation or overflow.

6. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

7. When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is SYNCHRONIZED.

8. This clause is hardware dependent and in addition to rules 1 thru 7, the implementor must specify how elementary items associated with this clause are handled regarding:

    a. The format on the external media of records or groups containing elementary items whose data description contains the SYNCHRONIZED clause.

    b. Any necessary generation of implicit FILLER, if the elementary item immediately preceding an item containing the SYNCHRONIZED clause does not terminate at an appropriate natural boundary. Such automatically generated FILLER positions are included in:

        (1) The size of any group to which the FILLER item belongs; and

        (2) The computer storage area allocation when the group item of which the FILLER item is a part appears as the object of a REDEFINES clause.

9. An implementor may, at his option, specify automatic alignment for some internal data formats; hence, implicitly SYNCHRONIZED data elements are not precluded. If the description of a data item includes the SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT clause, that clause takes precedence over any implicit synchronization for that item.

10. Any rules for synchronization of the records of a data file, as this effects the synchronization of elementary items, will be specified by the implementor.

6.41 THE TYPE CLAUSE

6.41.1 FUNCTION

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be generated.

6.41.2 GENERAL FORMAT

```
              ⎧  REPORT HEADING                                        ⎫
              ⎪  RH                                                    ⎪
              ⎪  PAGE HEADING                                          ⎪
              ⎪  PH                                                    ⎪
              ⎪  OVERFLOW HEADING                                      ⎪
              ⎪  OH                                                    ⎪
              ⎪ { CONTROL HEADING }        { identifier-n }            ⎪
              ⎪ { CH              }        { FINAL        }            ⎪
              ⎪                                                        ⎪
TYPE IS       ⎨  DETAIL                                                ⎬
              ⎪  DE                                                    ⎪
              ⎪ { CONTROL FOOTING }        { identifier-n }            ⎪
              ⎪ { CF              }        { FINAL        }            ⎪
              ⎪                                                        ⎪
              ⎪  OVERFLOW FOOTING                                      ⎪
              ⎪  OV                                                    ⎪
              ⎪  PAGE FOOTING                                          ⎪
              ⎪  PF                                                    ⎪
              ⎪  REPORT FOOTING                                        ⎪
              ⎩  RF                                                    ⎭
```

6.41.3 SYNTAX RULES

1. RH is an abbreviation for REPORT HEADING;
   PH is an abbreviation for PAGE HEADING;
   OH is an abbreviation for OVERFLOW HEADING;
   CH is an abbreviation for CONTROL HEADING;
   DE is an abbreviation for DETAIL;
   CF is an abbreviation for CONTROL FOOTING;
   OV is an abbreviation for OVERFLOW FOOTING;
   PF is an abbreviation for PAGE FOOTING;
   RF is an abbreviation for REPORT FOOTING.

6.41.4 GENERAL RULES

The level-number 01 identifies a particular report group to be generated as output and the TYPE clause in this entry indicates the time for generation of this report group. If the report group is described as other than TYPE DETAIL, its generation is an automatic report writer function. If the report group is described with the TYPE DETAIL clause the Procedure Division statement, GENERATE data-name directs the report writer to produce the named report group.

2. The REPORT HEADING entry indicates a report group that is produced only once at the beginning of a report during the execution of the first GENERATE statement. There can be only one report group of this type in a report. SOURCE clauses used in TYPE RH report groups refer to the values of data items at the time the first GENERATE statement is executed.

3. The PAGE HEADING entry indicates a report group that is produced at the beginning of each page according to PAGE and OVERFLOW condition rules as specified in Rule 19. There can be only one report group of this TYPE in a report.

4. The OVERFLOW HEADING entry indicates a report group that is produced at the beginning of a page following an OVERFLOW condition according to PAGE and OVERFLOW rules as specified in Rule 19. There can be only one report group of this TYPE in a report.

5. The CONTROL HEADING entry indicates a report group that is produced at the beginning of a control group for a designated identifier or, in the case of FINAL, is produced once before the first control group at the initiation of a report during the execution of the first GENERATE statement. There can be only one report group of this TYPE for each identifier and for the FINAL specified in a report. In order to produce any CONTROL HEADING report groups, a control break must occur. SOURCE clauses used in TYPE CONTROL HEADING FINAL report groups refer to the values of the items at the time the first GENERATE statement is executed.

6. The DETAIL entry indicates a report group that is produced for each GENERATE statement in the Procedure Division. Each DETAIL report group must have a unique data-name at the 01 level in a report.

7. The CONTROL FOOTING entry indicates a report group that is produced at the end of a control group for a designated identifier or is produced once at the termination of a report ending a FINAL control group. There can be only one report group of this TYPE for each identifier and for the FINAL entry specified in a report. In order to produce any CONTROL FOOTING report groups, a control break must occur. SOURCE clauses used in TYPE CONTROL FOOTING FINAL report groups refer to the values of the items at the time the TERMINATE statement is executed.

8. The OVERFLOW FOOTING indicates a report group that is produced at the bottom of a page following an OVERFLOW condition according to PAGE and OVERFLOW rules as specified in Rule 19. There can be only one report group of this TYPE in a report.

9. The PAGE FOOTING entry indicates a report group that is produced at the bottom of each page according to PAGE and OVERFLOW condition rules as specified in Rule 19. There can be only one report group of this TYPE in a report.

10. The REPORT FOOTING entry indicates a report group that is produced only once at the termination of a report. There can be only one report group of this type in a report. SOURCE clauses used in TYPE REPORT FOOTING report groups refer to the values of the items at the time the TERMINATE statement is executed.

11. Identifier, as well as FINAL, must be one of the identifiers described in the CONTROL clause in the Report Description entry.

12. A FINAL type control break may be designated only once for CONTROL HEADING or CONTROL FOOTING entries within a report.

13. Nothing precedes a REPORT HEADING entry and nothing follows a REPORT FOOTING entry within a report.

14. The HEADING or FOOTING report groups occur in the following Report Writer sequence, if all exist for a given report:

    REPORT HEADING (one occurrence only-first page)
    PAGE HEADING or OVERFLOW HEADING
    .
    .
    .
    CONTROL HEADING
    DETAIL
    CONTROL FOOTING
    .
    .
    .
    PAGE FOOTING or OVERFLOW FOOTING
    REPORT FOOTING (one occurrence only-last page)

15. CONTROL HEADING report groups are presented in the following hierarchical arrangement:

    Final Control Heading
    Major Control Heading
    .
    .
    .
    Minor Control Heading

    CONTROL FOOTING report groups are presented in the following hierarchical arrangement:

    Minor Control Footing
    .
    .
    .
    Major Control Footing
    Final Control Footing

16. CONTROL HEADING report groups appear with the current values of any indicated SOURCE data items before the DETAIL report groups of the CONTROL group are produced. CONTROL FOOTING report groups appear with the previous values of any indicated CONTROL SOURCE data items just after the DETAIL report groups of that CONTROL group have been produced. The USE procedures specified for a CONTROL FOOTING report group that refer to: a) SOURCE data items specified in the CONTROLS clause affect the previous value of the items; b) SOURCE data items not specified in the CONTROLS clause affect the current value of the items. These report groups appear whenever a control break is noted. LINE NUMBER determines the absolute or relative position of the CONTROL report groups exclusive of the other HEADING and FOOTING report groups.

17. The concept of the OVERFLOW condition in a Report Writer is based on the logical definition of a page format relative to the presentation of a complete control group. For purposes of the OVERFLOW condition, a complete control group depends on the change of a data item value within a designated order of specific data items. If the change is a minor control group break, the complete control group includes the HEADING, DETAIL and FOOTING report groups associated with the minor control specification. If the change is a major control group break, the complete control group includes the HEADING, DETAIL, and FOOTING report groups associated with the minor, intermediate, and major control specifications. Thus, during process time, if a page format does not allow a complete control group to be presented within the definition of the page, an OVERFLOW condition is said to exist from the last DETAIL report group printed in the control group on one page to the first

report group printed in the control group on the next page. Between the points of from and to described above, OVERFLOW FOOTING and OVERFLOW HEADING report groups may be produced, if specified. If a complete control group, as described above, and none of the next control group can be presented within the definition of the page, a PAGE condition is said to exist from the last DETAIL report group and therefore, PAGE FOOTING and PAGE HEADING report groups are produced, if specified.

18. PAGE HEADING and OVERFLOW HEADING, and PAGE FOOTING and OVERFLOW FOOTING clauses, if specified in a report, are mutually exclusive for any one page. The absence of a TYPE OVERFLOW HEADING clause indicates that TYPE PAGE HEADING report groups, if specified, are produced at the beginning of each page regardless of the condition that prompted the new page. Likewise, the absence of a TYPE OVERFLOW FOOTING indicates that TYPE PAGE FOOTING report groups, if specified, are produced at the bottom of each page regardless of the condition that ended the current page.

19. In order to recognize the OVERFLOW condition within the Report Writer and to determine the difference between an OVERFLOW condition and a PAGE condition, the PAGE LIMITS clause must be given including an explicit LAST DETAIL clause. If both TYPE PAGE HEADING and OVERFLOW HEADING or TYPE PAGE FOOTING and OVERFLOW FOOTING report groups are specified in the same report and if the line counter will exceed the LAST DETAIL limit for generation of the current report group, the following rules apply:

    a. Without an explicit PAGE LIMITS FOOTING clause, if the current report group is not the first report group of a new control group, an OVERFLOW condition exists from this position on the page to the position on the next page where the FIRST DETAIL report group can be presented. If the current report group is the first report group of a new control group, a PAGE condition exists. TYPE CONTROL FOOTING report groups are considered part of the last control group. TYPE CONTROL HEADING report groups are considered part of the next or current control group.

    b. With an explicit PAGE LIMITS FOOTING clause, if the current report group is a TYPE DETAIL report group, an OVERFLOW condition exists as stated in a. above. If the current report group is a CONTROL FOOTING report group, an additional test is made to determine if the LINE-COUNTER will exceed the FOOTING limit for generation of the complete CONTROL FOOTING report group. If all the report groups associated with this control break can be produced within the limit specified, a PAGE condition exists following the CONTROL FOOTING report group. If all the report groups associated with this control break cannot be produced within the limit specified, an OVERFLOW condition exists, which means the TYPE CONTROL FOOTING report groups are produced on the following page.

c. Without an explicit PAGE LIMITS LAST DETAIL clause, an OVERFLOW condition cannot exist within the Report Writer. Therefore, with or without the PAGE LIMITS FOOTING clause, TYPE PAGE FOOTING, as differentiated from TYPE OVERFLOW FOOTING, are the only report groups that are produced, if specified, after TYPE DETAIL and TYPE CONTROL report groups on a page.

d. The rules stated in Rule 18 above apply regardless of the conditions that may be recognized by the Report Writer as described in Rule 19.

6.42 THE USAGE CLAUSE

6.42.1 FUNCTION

The USAGE clause specifies the format of a data item in the computer storage.

6.42.2 GENERAL FORMAT

```
              ⎧ COMPUTATIONAL   ⎫
              ⎪ COMP            ⎪
              ⎪ COMPUTATIONAL-n ⎪
[USAGE IS]    ⎨ COMP-n          ⎬
              ⎪ DISPLAY         ⎪
              ⎪ DISPLAY-n       ⎪
              ⎪ INDEX           ⎪
              ⎩ INDEX-n         ⎭
```

6.42.3 SYNTAX RULES

1. The PICTURE of a COMPUTATIONAL item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', one or more 'P's, and the insertion character '0' (see 6.31, The PICTURE Clause).

2. The USAGE IS DISPLAY clause indicates that the format of the data is a Standard Data Format.

3. The USAGE clause for a report group item can only specify USAGE IS DISPLAY.

4. COMP is an abbreviation for COMPUTATIONAL;
   COMP-n is an abbreviation for COMPUTATIONAL-n.

6.42.4 GENERAL RULES

1. The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

2. This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

3. Suffixing a USAGE clause with a hyphen, followed by a single digit number (1-9), allows for several variations to a general type of USAGE. Any implementor who provides such variations (binary, decimal, and floating-point computations, etc.) assigns the variations in the order of their preference (that is,

efficiency or frequency of use). If no suffix is written, or if the integer given in the suffix is unassigned, the recommended (first) variation is used.

4. A COMPUTATIONAL item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group item itself is not COMPUTATIONAL (cannot be used in computations). If the USAGE clause specified at the group level is suffixed, the elementary items are considered to be the same, and may not have a contradicting form of computational usage specified.

5. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is assumed to be DISPLAY.

6. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table-element. The elementary item cannot be a conditional variable. The method of representation and the actual value assigned are determined by the implementor. If a group item is described with the USAGE IS INDEX clause the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in the SEARCH or SET statement or in a relation condition.

7. An index data item can be referred to directly only in a SEARCH or SET statement or in a relation condition. An index data item can be part of a group which is referred to in a MOVE or input-output statement, in which case no conversion will take place.

8. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause. The external and internal format of an index data item is specified by the implementor.

9. USAGE IS DISPLAY-n permits the implementor to specify variations of different character representations for those hardware units having this capability.

6.43 THE VALUE CLAUSE

6.43.1 FUNCTION

The VALUE clause defines the value of constants, the initial value of working-storage items, or the values associated with a condition-name.

6.43.2 GENERAL FORMAT

Format 1

$$\underline{\text{VALUE}}\ \text{IS literal}$$

Format 2

$$\left\{\begin{matrix}\underline{\text{VALUE}}\ \text{IS}\\ \underline{\text{VALUES}}\ \text{ARE}\end{matrix}\right\}\ \text{literal-1}\left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\text{literal-2}\right]$$
$$\left[,\ \text{literal-3}\left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\text{literal-4}\right]\right]\ldots$$

6.43.3 SYNTAX RULES

1. The words THRU and THROUGH are equivalent.

6.43.4 GENERAL RULES

1. The VALUE clause cannot be stated for any item whose size, explicitly or implicitly, is variable.

2. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:

   a. If the category of the item is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a working-storage item, the literal is aligned according to the alignment rules except that the literal must not have a value which would require truncation of non-zero digits. A negative numeric literal must be associated with a signed numeric (S9) PICTURE character-string.

b. If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned according to the alignment rules (see 6.23, The JUSTIFIED Clause), except that the number of characters in the literal must not exceed the size of the item.

c. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, for example, for PICTURE PPP99, the literal must be within the range .00000 thru .00099.

d. The function of the BLANK WHEN ZERO clause or any editing characters in a PICTURE clause has no effect on initialization of the item. The VALUE clause is the only clause that may (depending on its usage) provide initialization. Editing characters are included however, in determining the size of the item. Therefore, the VALUE for an edited item must be presented in an edited form.

3. A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

6.43.5 CONDITION-NAME RULES

1. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

2. Format 2 can be used only in connection with condition-names (see paragraph 3.2.1.2.2, Condition-Name). Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

6.43.6 DATA DESCRIPTION ENTRIES OTHER THAN CONDITION-NAMES

1. Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

   a. In the File Section, the VALUE clause may be used only in condition-name entries.

   b. In the Working-Storage Section, the VALUE clause must be used in condition-name entries, and it may also be used to specify the initial value of any data item. It causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item's description, the initial value is undefined.

c. In the Constant Section, the VALUE clause must be used to specify the value assumed by each constant data item in the object program.

d. In the Linkage Section, the VALUE clause may be used only in condition-name entries.

e. In the Report Section, the VALUE clause causes the report data item to assume the specified value each time its report group is presented. This clause may be used only at the elementary level in the Report Section.

2. The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

3. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

4. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a non-numeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

5. The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED OR USAGE (other than USAGE IS DISPLAY).

6. Within a given record description the VALUE clause must not be stated in a data description entry that is subsequent to a data description entry in which an OCCURS clause with a DEPENDING ON phrase appears.

6.44 THE VALUE OF CLAUSE

6.44.1 FUNCTION

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

6.44.2 GENERAL FORMAT

```
VALUE OF  data-name-1  IS  { literal-1   }
                           { data-name-2 }
      [ , data-name-3  IS  { literal-2   } ] ...
                           { data-name-4 }
```

6.44.3 SYNTAX RULES

1. Data-name-1, data-name-2, data-name-3, etc., should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

6.44.4 GENERAL RULES

1. Each data-name-1, data-name-3, etc., must be in one of the label records; data-name-2, data-name-4, etc., must be in the Working-Storage or Constant Section. For an:

   a. Input File: The appropriate label routine checks to see if the value of data-name-1 is equal to the value of literal-1, or of data-name-2, whichever has been specified.

   b. Output File: At the appropriate time the value of data-name-1 is made equal to the value of literal-1, or of a data-name-2, whichever has been specified.

2. A figurative constant may be substituted in the format above wherever a literal is specified.

3. If label records are standard (see 6.24, The LABEL RECORDS Clause), then data-name-1, data-name-3, etc., must be fixed names supplied by the individual implementors.

CHAPTER 7

THE PROCEDURE DIVISION

7.1 GENERAL DESCRIPTION

The Procedure Division must be included in every COBOL source program. This division contains declaratives and procedures.

7.1.1 DECLARATIVES

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES.

There are two statements that are called declarative statements: USE and COPY (see 7.41, The USE Statement; 9.2, The COPY Statement).

7.1.2 PROCEDURES

A procedure is composed of a paragraph, or group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the Source Program in which it occurs. It consists of a paragraph-name (which may be qualified), or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear.

A section consists of a section header followed by one or more successive paragraphs. A section ends immediately before the next section or at the end of the Procedure Division or, in the Declaratives portion of the Procedure Division, at the key words END DECLARATIVES.

A paragraph consists of a paragraph-name followed by one or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the Procedure Division or, in the Declaratives portion of the Procedure Division, at the key words END DECLARATIVES.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term 'identifier' is defined as the word or words necessary to make unique reference to a data item.

7.1.3 EXECUTION

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules in this chapter indicate some other order.

7.1.4 PROCEDURE DIVISION STRUCTURE

7.1.4.1 Procedure Division Header

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION   [ USING identifier-1  [, identifier-2] ... ].
```

The USING clause is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING clause.

Each of the operands in the USING clause of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

When the USING clause is present, the object program operates as if each occurrence of identifier-1, identifier-2, etc., in the Procedure Division had been replaced by the corresponding identifier from the USING clause in the CALL statement of the calling program. That is, the corresponding identifiers refer to a single set of data which is available to both the called and calling programs. The correspondence is positional and not by name. An identifier must not appear more than once in the same USING clause.

7.1.4.2 Procedure Division Body

The body of the Procedure Division must conform to one of the following formats:

Format 1

```
[DECLARATIVES.
 {section-name SECTION.  declarative-sentence
 {paragraph-name. { sentence } ... } ... } ...
 END DECLARATIVES.]

 {section-name SECTION [priority-number].
 {paragraph-name. { sentence } ... } ...} ...
```

Format 2

```
{ paragraph-name. { sentence } ... }...
```

7.2 STATEMENTS AND SENTENCES

There are three types of statements: imperative statements, conditional statements, and compiler directing statements.

There are three types of sentences: imperative sentences, conditional sentences, and compiler directing sentences.

7.2.1 CONDITIONAL STATEMENTS AND CONDITIONAL SENTENCES

7.2.1.1 Definition of Conditional Statement

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

A conditional statement is an IF, READ, SEARCH, or RETURN statement, or a WRITE statement that specifies an INVALID KEY or END-OF-PAGE phrase, or an arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the optional phrase SIZE ERROR.

7.2.1.2 Definition of Conditional Sentence

A conditional sentence is a conditional statement optionally preceded by an imperative statement terminated by a period followed by a space.

7.2.2 COMPILER DIRECTING STATEMENTS AND COMPILER DIRECTING SENTENCES

7.2.2.1 Definition of Compiler Directing Statement

A compiler directing statement consists of a compiler directing verb and its operands. The compiler directing verbs are COPY and ENTER (see 9.2, The COPY Statement; 7.17, The ENTER Statement). A compiler directing statement causes the compiler to take a specific action during compilation.

7.2.2.2 Definition of Compiler Directing Sentence

A compiler directing sentence is a single compiler directing statement terminated by a period followed by a space.

7.2.3 IMPERATIVE STATEMENTS AND IMPERATIVE SENTENCES

7.2.3.1 Definition of Imperative Statement

An imperative statement indicates a specific action to be taken by the object program.

An imperative statement is any statement that is neither a conditional statement, nor a compiler directing statement, nor a USE statement. An imperative statement may consist of a sequence of imperative statements each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT
ADD*
ALTER
CALL
CANCEL
CLOSE
COMPUTE*
DISPLAY
DIVIDE*
EXAMINE
EXIT
GENERATE
GO
HOLD
INITIATE
MOVE
MULTIPLY*
OPEN
PERFORM
PROCESS
RELEASE
SEEK
SET
SORT
STOP
SUBTRACT*
SUSPEND
TERMINATE
WRITE**

* Without the optional phrase SIZE ERROR.
** Without the optional phrase INVALID KEY or END-OF-PAGE.

Whenever an imperative statement appears in the General Format of statements described in this chapter, the imperative statement refers to that sequence of consecutive imperative statements ended by a period or an ELSE associated with a previous IF verb or a WHEN associated with a previous SEARCH verb.

7.2.3.2 Definition of Imperative Sentence

An imperative sentence is an imperative statement terminated by a period followed by a space.

7.3 ARITHMETIC EXPRESSIONS

7.3.1 DEFINITION OF AN ARITHMETIC EXPRESSION

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, (identifiers or numeric literals), arithmetic operators and parentheses are given in Figure 7-1, Combination of Symbols in Arithmetic Expressions.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

7.3.2 ARITHMETIC OPERATORS

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a space and followed by a space.

Binary Arithmetic Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operator	Meaning
+	The effect of multiplication by the numeric literal +1
-	The effect of multiplication by the numeric literal -1

7.3.3 FORMATION AND EVALUATION RULES

1. Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. When parentheses are used, a space may appear between the left parenthesis and the left-most element and between the right parenthesis and the right-most element, if desired. Expressions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

1st - Unary plus and minus
2nd - Exponentiation
3rd - Multiplication and Division
4th - Addition and Subtraction

2. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in the table below (Figure 7-1), where:
   a. The letter 'P' indicates a permissible pair of symbols.
   b. The character '-' represents an invalid pair.
   c. Variable represents an identifier or literal.

FIRST SYMBOL	SECOND SYMBOL				
	VARIABLE	*/** +-	Unary + or -	(	)
VARIABLE	-	P	-	-	P
*/** + -	P	-	P	P	-
Unary + or -	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

Figure 7-1. Combination of Symbols in Arithmetic Expressions.

4. An arithmetic expression may only begin with the symbol '(', '+', '-', or a variable and may only end with a ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

5. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items. See, for example, 7.8.3.3. Each implementor will indicate the techniques used in handling arithmetic expressions.

7.4 CONDITIONS

7.4.1 GENERAL DESCRIPTION

A condition enables the object program to select between alternate paths of control depending upon the truth value of a test.

A condition is one of the following:

relation condition
class condition
condition-name condition
switch-status condition
sign condition
$\underline{\text{NOT}}$ condition
$\text{condition} \quad \left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\} \quad \text{condition} \quad \left[\ \left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\} \quad \text{condition}\ \right] \ldots$

Any condition may be enclosed in parentheses. The truth value of a parenthesized condition is determined from the evaluation of the truth values of its constituents. A parenthesized condition is a condition in the sense of the last two items of the preceding list.

7.4.2 RELATION CONDITION

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, or an arithmetic expression. Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

The format for a relation condition is as follows:

$$\underline{\text{IF}} \left\{ \begin{array}{l} \text{identifier-1} \\ \\ \text{literal-1} \\ \\ \text{arithmetic-} \\ \quad\text{expression-1} \end{array} \right\} \left\{ \begin{array}{l} \text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{GREATER}}\text{ THAN} \\ \text{IS [}\underline{\text{NOT}}\text{] } > \\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{LESS}}\text{ THAN} \\ \text{IS [}\underline{\text{NOT}}\text{] } < \\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{EQUAL}}\text{ }\underline{\text{TO}} \\ \text{IS [}\underline{\text{NOT}}\text{] } = \\ \text{IS }\underline{\text{UNEQUAL}}\text{ }\underline{\text{TO}} \\ \underline{\text{EQUALS}} \\ \underline{\text{EXCEEDS}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-2} \\ \\ \text{literal-2} \\ \\ \text{arithmetic-} \\ \quad\text{expression-2} \end{array} \right\}$$

The word 'IF' is not part of the condition, but is shown in the above format to improve clarity.

The first operand (identifier-1, literal-1, or arithmetic-expression-1) is called the subject of the condition; the second operand (identifier-2, literal-2, arithmetic-expression-2) is called the object of the condition. The subject and the object may not both be literals.

The relational operators specify the type of comparison to be made in a relation condition. The relational operators must be preceded by a space and followed by a space. The meaning of the relational operators is as follows:

Meaning	Relational Operator
Greater than or not greater than	IS [NOT] GREATER THAN IS [NOT] >
Less than or not less than	IS [NOT] LESS THAN IS [NOT] <
Equal to or not equal to	IS [NOT] EQUAL TO IS [NOT] =
Not equal to	IS UNEQUAL TO
Equal to	EQUALS
Greater than	EXCEEDS

NOTE: In the formats above, the required relational characters '<', '>', and '=' are not underlined to avoid confusion with other symbols such as $\underline{>}$ (greater than or equal to).

7.4.2.1 Comparison of Numeric Operands

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands. The length of the operands, in terms of number of digits, is not significant. Zero is considered a unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

7.4.2.2 Comparison of Nonnumeric Operands

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters.

The size of an operand is the total number of characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same, implicitly or explicitly.

There are two cases to consider: operands of equal size and operands of unequal size.

1. Operands of Equal Size

   If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the item is reached, whichever comes first. The items are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

   The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

2. Operands of Unequal Size

   If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size (see 7.4.2.2.1, Operands of Equal Size).

7.4.2.3 Comparisons Involving Index-Names and/or Index Data Items

Full relation tests may be made between:

1. Two index-names. The result is the same as if the corresponding occurrence numbers are compared.
2. An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.
3. An index data item and an index name or another index data item. The actual values are compared without conversion.
4. The result of the comparison of an index data item with any data item or literal not specified above is undefined.

7.4.3 CLASS CONDITION

The class condition determines whether the operand is numeric, that is, consists entirely of the characters '0', '1', '2', '3', ..., '9', with or without an operational sign, or alphabetic, that is, consists entirely of the characters 'A', 'B', 'C', ..., 'Z', space. The general format for the class condition is as follows:

$$\underline{\text{IF}} \quad \text{identifier} \quad \text{IS} \quad [\underline{\text{NOT}}] \quad \left\{ \begin{array}{l} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \end{array} \right\}$$

The word 'IF' is not part of the condition, but is shown in the above format to improve clarity.

The usage of the operand being tested must be described, implicitly or explicitly, as display.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic. If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space.

7.4.4 CONDITION-NAME CONDITION (CONDITIONAL VARIABLE)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name. The general format for the condition-name condition is as follows:

```
IF              condition-name
```

The word 'IF' is not part of the condition, but is shown in the above format to improve clarity.

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

7.4.5 SWITCH-STATUS CONDITION

A switch-status condition determines the on or off status of an implementor defined switch. The implementor-name and its associated ON or OFF value must be named in the SPECIAL-NAMES paragraph of the Environment Division. The general format for the switch-status condition is as follows:

```
IF              condition-name
```

The word 'IF' is not part of the condition, but is shown in the above format to improve clarity.

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

7.4.6 SIGN CONDITION

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero. The general format for a sign condition is as follows:

$$\underline{\text{IF}} \quad \left\{\begin{array}{l}\text{identifier}\\ \\ \text{arithmetic-expression}\end{array}\right\} \quad \text{IS } [\ \underline{\text{NOT}}\ ] \quad \left\{\begin{array}{l}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{array}\right\}$$

The word 'IF' is not part of the condition, but is shown in the above format to improve clarity.

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

7.4.7 COMPOUND CONDITIONS

7.4.7.1 Combined Conditions

Conditions may be combined by logical operators, according to specified rules, to form compound conditions. The logical operators, AND, OR, and NOT, must be preceded by a space and followed by a space. The meaning of the logical operators is as follows:

Logical Operator	Meaning
OR	Logical Inclusive Or
AND	Logical Conjunction
NOT	Logical Negation

Figure 7-2 indicates the relationship between the logical operators where A and B each represent a condition.

Condition		Condition and Value		
A	B	A AND B	A OR B	NOT A
True	True	True	True	False
False	True	False	True	True
True	False	False	True	False
False	False	False	False	True

Figure 7-2. Relationship of Conditions, Logical Operators, and Truth Values

The general format of a combined condition is:

$$\text{IF} \quad \text{condition} \quad \left\{\begin{array}{c}\underline{\text{AND}}\\ \underline{\text{OR}}\end{array}\right\} \quad \text{condition} \quad \left[\quad \left\{\begin{array}{c}\underline{\text{AND}}\\ \underline{\text{OR}}\end{array}\right\} \quad \text{condition} \quad \right] \quad \ldots$$

The word 'IF' is not part of the condition, but is shown in the above format to improve clarity.

Figure 7-3 indicates the ways in which conditions and logical operators may be combined.

FIRST SYMBOL	SECOND SYMBOL					
	Condition	OR	AND	NOT	(	)
Condition	-	P	P	-	-	P
OR	P	-	-	P	P	-
AND	P	-	-	P	P	-
NOT	P*	-	-	-	P	-
(	P	-	-	P	P	-
)	-	P	P	-	-	P

Figure 7-3. Combinations of Conditions and logical operators.

Notes: 'P' indicates that the pair is permissible, and the '-' indicates a symbol pair that is not permissible. Thus, the pair 'OR NOT' is permissible, while the pair 'NOT OR' is not permissible.

* Permissible only if the condition is not itself a 'not condition'.

7.4.7.2 Abbreviated Combined Relation Conditions

When relation conditions are written in a consecutive sequence, any relation condition except the first may be abbreviated by:

(1) The omission of the subject of the relation condition, or

(2) The omission of the subject and relational operator of the relation condition.

Within a sequence of relation conditions both forms of abbreviation may be used, the effect of using such abbreviations is as if the omitted subject was replaced by the last preceding stated subject or the omitted relational operator was replaced by the last preceding stated relational operator.

Ambiguity may result from using 'NOT' in conjunction with abbreviations. In this event NOT will be interpreted as a logical operator rather than as part of a relational operator. Thus:

a > b AND NOT > c OR d

is equivalent to:

a > b AND NOT a > c OR a > d

or a > b AND (NOT a > c) OR a > d.

7.4.8 EVALUATION RULES

The evaluation rules for conditions are analogous to those given for arithmetic expressions (see 7.3.3, Formation and Evaluation Rules) except that the following hierarchy applies:

arithmetic expression
all relational operators
NOT
AND
OR

The arithmetic expressions are evaluated as described in 7.3.3, Formation and Evaluation Rules; the relational operators are evaluated as described in 7.4.2, Relation Condition; and the logical operators are evaluated according to Figure 7-2, Relationship of Conditions, Logical Operators, and Truth Values.

7.5 CATEGORIES OF STATEMENTS

Category	Verbs
Arithmetic	ADD COMPUTE DIVIDE EXAMINE (TALLYING) MULTIPLY SUBTRACT
Asynchronous Processing	HOLD PROCESS
Compiler Directing	COPY ENTER
Compiler Directing Declarative	COPY USE
Conditional	ADD (SIZE ERROR) COMPUTE (SIZE ERROR) DIVIDE (SIZE ERROR) GO TO (DEPENDING) IF MULTIPLY (SIZE ERROR) READ (END) RETURN (END) SEARCH SUBTRACT (SIZE ERROR) WRITE INVALID KEY or END-OF-PAGE
Data Movement	EXAMINE (REPLACING) MOVE
Ending	STOP
Input-Output	ACCEPT CLOSE DISPLAY OPEN READ SEEK STOP (literal) SUSPEND WRITE

Category	**Verbs**
Inter-Program Communicating	CALL CANCEL
Procedure Branching	ALTER CALL EXIT GO TO PERFORM
Report Writing	GENERATE INITIATE TERMINATE
Sorting	RELEASE RETURN SORT
Table Handling	SEARCH SET

IF is a verb in the COBOL sense; it is recognized that it is not a verb in English.

7.5.1 SPECIFIC STATEMENT FORMATS

The specific statement formats, together with a detailed discussion of the restrictions and limitations associated with each, appear beginning in paragraph 7.7, in alphabetic sequence.

7.6 COMMON OPTIONS IN STATEMENT FORMATS

In the statement descriptions that follow, several options appear frequently: the ROUNDED option, the SIZE ERROR option, and the CORRESPONDING option.

In the discussion below, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

7.6.1 THE ROUNDED OPTION

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant-identifier are represented by the character 'P' in the picture for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

7.6.2 THE SIZE ERROR OPTION

If, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified.

1. If the SIZE ERROR option is not specified and a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

2. If the SIZE ERROR option is specified and a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR option is executed.

   For ADD and SUBTRACT CORRESPONDING, if any of the individual operations produce a size error condition, the imperative statement in the SIZE ERROR clause is not executed until all of the individual additions or subtractions are completed.

7.6.3 THE CORRESPONDING OPTION

For the purpose of this discussion, d_1 and d_2 must each be identifiers that refer to group items. A pair of data items, one from d_1 and one from d_2 correspond if the following conditions exist:

1. A data item in d_1 and a data item in d_2 have the same name and the same qualification up to, but not including, d_1 and d_2.

2. At least one of the data items is an elementary data item in the case of a MOVE statement with the CORRESPONDING option; and both of the data items are elementary numeric data items in the case of

$$\left\{\begin{array}{l}\underline{\text{ADD}}\\ \underline{\text{SUBTRACT}}\end{array}\right\} \qquad \underline{\text{CORRESPONDING}}$$

3. Neither d_1 or d_2 may be data items with level-number 66, 77, or 88 nor be described with the USAGE IS INDEX clause.

4. A data item that is subordinate to d_1 or d_2 and contains a REDEFINES, OCCURS or USAGE IS INDEX clause is ignored, as well as these data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, d_1 and d_2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

7.6.4 THE ARITHMETIC STATEMENTS

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

2. The maximum size of each operand is eighteen (18) decimal digits.

7.6.5 OVERLAPPING OPERANDS

When a sending and a receiving item in an arithmetic statement or an EXAMINE, MOVE or SET statement share a part of their storage areas, the result of the execution of such a statement is undefined.

7.6.6 MULTIPLE RESULTS IN ARITHMETIC STATEMENTS

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written in the following way:

1. A statement which performs all arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence that the multiple results are listed.

   The result of the statement

   ```
   ADD a, b, c TO c, d (c), e
   ```

   is equivalent to

   ```
   ADD a, b, c   GIVING temp
   ADD temp TO c
   ADD temp TO d (c)
   ADD temp TO e
   ```

   where 'temp' is an intermediate result item provided by the implementor.

7.7 THE ACCEPT STATEMENT

7.7.1 FUNCTION

The ACCEPT statement causes low volume data to be made available to the specified data-name.

7.7.2 GENERAL FORMAT

ACCEPT identifier [FROM mnemonic-name]

7.7.3 SYNTAX RULES

1. The hardware device must be associated with the mnemonic-name in the SPECIAL-NAMES paragraph of the Environment Division.

7.7.4 GENERAL RULES

1. The ACCEPT statement causes the transfer of data from the hardware device. This data replaces the contents of the data item named by the identifier.

2. The implementor will define, for each hardware device, the size of a data transfer.

3. If a hardware device is capable of transferring data of the same size as the receiving data item, the transferred data is stored in the receiving data item.

4. If a hardware device is not capable of transferring data of the same size as the receiving data item, then:
   a. If the size of the receiving data item (or of the portion of the receiving data item not yet currently occupied by transferred data) exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item (or the portion of the receiving data item not yet occupied), and additional data is requested.

b. If the size of the transferred data exceeds the size of the receiving data item (or of the portion of the receiving data item not yet occupied by transferred data), only the left-most characters of the transferred data are stored in the receiving data item (or in the portion remaining).

5. If the FROM option is not given, the device that the implementor specifies as standard is used.

7.8 THE ADD STATEMENT

7.8.1 FUNCTION

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

7.8.2 GENERAL FORMAT

Format 1

$$\underline{\text{ADD}} \quad \left\{\begin{matrix}\text{literal-1}\\ \text{identifier-1}\end{matrix}\right\} \quad \left[\begin{matrix}\text{, literal-2}\\ \text{, identifier-2}\end{matrix}\right] \quad \ldots \quad \underline{\text{TO}}\ \text{identifier-m}\ [\underline{\text{ROUNDED}}]$$

$$[\text{, identifier-n}\ [\underline{\text{ROUNDED}}]]\ \ldots\ [\text{; ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

Format 2

$$\underline{\text{ADD}} \quad \left\{\begin{matrix}\text{literal-1}\\ \text{identifier-1}\end{matrix}\right\} \quad \left\{\begin{matrix}\text{, literal-2}\\ \text{, identifier-2}\end{matrix}\right\} \quad \left[\begin{matrix}\text{, literal-3}\\ \text{, identifier-3}\end{matrix}\right] \quad \ldots$$

$$\underline{\text{GIVING}} \quad \text{identifier-m}\ [\underline{\text{ROUNDED}}]$$

$$[\text{, identifier-n}\ [\underline{\text{ROUNDED}}]]\ \ldots\ [\text{; ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

Format 3

$$\underline{\text{ADD}} \quad \left\{\begin{matrix}\underline{\text{CORR}}\\ \underline{\text{CORRESPONDING}}\end{matrix}\right\} \quad \text{identifier-1}\ \underline{\text{TO}}\ \text{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[\text{; ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

7.8.3 SYNTAX RULES

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that identifiers appearing only to the right of the word GIVING may refer to data items that contain editing symbols.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data items that follow the word GIVING, aligned on their decimal points, must not contain more than eighteen digits.

4. CORR is an abbreviation for CORRESPONDING.

7.8.4 GENERAL RULES

1. See 7.6.1, the ROUNDED Option; 7.6.2, The SIZE ERROR Option; 7.6.3, the CORRESPONDING option; 7.6.6, Multiple Results in Arithmetic Statements; and 3.2.1.2.5, Special Registers.

2. If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value of each identifier-m, identifier-n, ..., and the result is stored in each resultant-identifier, identifier-m, ..., respectively.

3. If Format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of each identifier-m, identifier-n, ..., the resultant-identifiers.

4. If Format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

5. The compiler insures that enough places are carried so as not to lose any significant digits during execution.

7.9 THE ALTER STATEMENT

7.9.1 FUNCTION

The ALTER statement modifies a predetermined sequence of operations.

7.9.2 GENERAL FORMAT

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2

      [, procedure-name-3 TO [PROCEED TO] procedure-name-4] ...
```

7.9.3 SYNTAX RULES

1. Each procedure-name-1, procedure-name-3, ..., is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING option.

2. Each procedure-name-2, procedure-name-4, ..., is the name of a paragraph or section in the Procedure Division.

7.9.4 GENERAL RULES

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ..., so that subsequent executions of the modified GO TO statements causes transfer of control to procedure-name-2, procedure-name-4, ..., respectively. Modified GO TO statements in independent segments may, under some circumstances, be returned to their initial states (see 8.1.2.3, Independent Segments).

2. A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority.

   All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in an overlayable fixed segment. See Chapter 8, Segmentation.

7.10 THE CALL STATEMENT

7.10.1 FUNCTION

The CALL statement causes control to be transferred from one object program to another, within the run unit.

7.10.2 GENERAL FORMAT

$$\underline{\text{CALL}} \quad \left\{ \begin{array}{l} \text{literal-1} \\ \\ \text{identifier-1} \end{array} \right\}$$

$$\left[\underline{\text{USING}} \text{ identifier-2 } \left[\text{, identifier-3} \right] \ldots \right]$$

7.10.3 SYNTAX RULES

1. Literal-1 must be a nonnumeric literal.
2. Identifier-1 must be defined such that its value can be a program-name.
3. The USING clause is included in the CALL statement only if there is a USING clause in the Procedure Division header of the called program and the number of operands in each USING clause must be identical.

7.10.4 GENERAL RULES

1. The program whose name is specified by the value of literal-1 or identifier-1 is the called program; the program in which the CALL statement appears is the calling program.
2. The execution of a CALL statement causes control to pass to the called program.
3. The state of a called program is undefined.
4. Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.
5. Each of the operands in the USING clause must have been defined as a data item in the File Section, Working-Storage Section, Constant Section, or Linkage Section, and must have a level-number of 01 or 77.

6. The identifiers, specified by the USING clause of the CALL statement, indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the identifiers in the USING clause of the CALL statement and the USING clause in the Procedure Division header is critical. Corresponding identifiers refer to a single set of data which is available to the called and calling program. The correspondence is positional, not by name.

7.11 THE CANCEL STATEMENT

7.11.1 FUNCTION

The CANCEL statement releases the memory areas occupied by the named program.

7.11.2 GENERAL FORMAT

$$\underline{\text{CANCEL}} \quad \left\{ \begin{array}{l} \text{literal-1} \\ \text{identifier-1} \end{array} \right\} \quad \left[\begin{array}{ll} , & \text{literal-2} \\ , & \text{identifier-2} \end{array} \right] \ldots$$

7.11.3 SYNTAX RULES

1. Literal-1, literal-2, ..., must be a nonnumeric literal.

2. Identifier-1, identifier-2, ..., must each be defined such that its value can be a program-name.

7.11.4 GENERAL RULES

1. Subsequent to the execution of a CANCEL statement, the program named therein ceases to have any logical relationship to the program in which the CANCEL statement appears.

2. The programs named in the CANCEL statement must be named in CALL statements within the same program in which the CANCEL statement appears.

3. A logical relationship to a cancelled subprogram is established only by execution of a subsequent CALL statement.

4. A called program is cancelled either by being directly named as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

7.12 THE CLOSE STATEMENT

7.12.1 FUNCTION

The CLOSE statement terminates the processing of reels, units, and files with optional rewind and/or lock where applicable.

7.12.2 GENERAL FORMAT

$$\underline{\text{CLOSE}}\ \text{file-name-1}\ \left[\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right]\left[\ \text{WITH}\ \left\{\begin{array}{l}\underline{\text{NO REWIND}}\\ \underline{\text{LOCK}}\end{array}\right\}\right]$$

$$\left[\ ,\ \text{file-name-2}\ \left[\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right]\ \left[\ \text{WITH}\ \left\{\begin{array}{l}\underline{\text{NO REWIND}}\\ \underline{\text{LOCK}}\end{array}\right\}\right]\right]\ \ldots$$

7.12.3 SYNTAX RULES

1. Each file-name is the name of a file upon which the CLOSE statement is to operate; it must not be the name of a sort-file.

2. The REEL and WITH NO REWIND options apply only to files stored on tape devices and other devices to which these terms are applicable. The UNIT option is only applicable to mass storage files in the sequential access mode.

7.12.4 GENERAL RULES

In the discussion below, the term 'unit' applies to all input-output devices; the term 'reel' applies to tape devices. Treatment of mass storage devices in the sequential access mode is logically equivalent to the treatment of a file on tape or analogous media.

1. For the purpose of showing the effect of various CLOSE options as applied to various storage media, all input and output and input-output files are divided into the following categories.

   a. Non-reel. A file whose input or output medium is such that the concepts of rewinding and reels have no meaning.

   b. Sequential single reel/unit. A sequential file that is entirely contained on one unit.

   c. Sequential multi-reel/unit. A sequential file that is contained on more than one unit.

d. Random single-unit. A file in the random access mode that is entirely contained on a single mass storage unit.

e. Random multi-unit. A file in the random access mode that may be contained on more than one mass storage unit.

2. The results of executing each CLOSE option for each type of file are summarized in Figure 7-5. The definitions of the symbols in the Figure are given below. Where the definition depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Previous Reels/Units Unaffected

Input files and Input-Output files:

All reels/units in the file prior to the current reel/unit are processed according to the implementor's standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement. If the current reel/unit is not the last in the file, the reels/units in the file following the current one are not processed in any way.

Output files:

All reels/units in the file prior to the current reel/unit are processed according to the implementor's standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement.

B. No Rewind Of Current Reel

The current reel/unit is left in its current position.

C. Standard Close File

Input files and Input-Output files (Sequential Access Mode):

If the file is positioned at its end and a label record is specified for the file, the label is processed according to the implementor's standard label convention. The behavior of the CLOSE statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. In addition, other closing operations specified by the implementor are executed. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed. If the file is positioned other than at its end, the closing operations specified by the implementor are executed, but there is no ending label processing. An input file, or an input-output file, is considered to be at the end of the file if the imperative statement in the AT END phrase of the READ statement has been executed and no CLOSE statement has been executed.

Input files and Input-Output files (Random Access Mode); Output files (Random or Sequential Access Mode):

If a label record is specified for the file, the label is processed according to the implementor's standard label convention. The behavior of the CLOSE statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. In addition, other closing operations specified by the implementor are executed. If label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed.

D. Standard Reel/Unit Lock

An appropriate technique is supplied to insure that the current reel or unit cannot be processed again as a part of this file. (The current reel is rewound.)

E. Standard File Lock

An appropriate technique is supplied to insure that this file cannot be opened again during this execution of this object program.

F. Standard Close Reel/Unit

Input Files:

The following operations are executed:

1. A reel/unit swap.
2. The standard beginning reel/unit label procedure and the user's beginning reel/unit label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.
3. Makes available the next data record on the new reel or the next mass storage record.

Output files and Input-Output files:

The following operations are executed:

1. (For output files only) The standard ending reel/unit label procedure and the user's ending reel/unit label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

2. A reel/unit swap.

3. The standard beginning reel/unit label procedure and the user's beginning reel/unit label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

4. (For input-output files only) Makes the next data record on the next mass storage unit available.

G. Rewind

The current reel or analogous device is positioned at the physical beginning of its content.

X. Illegal

This is an illegal combination of a CLOSE option and a file type. The results at object time may be unpredictable.

3. The action taken if a file that has been opened and is not closed prior to the execution of the STOP RUN statement is specified by the implementor.

4. If the file has been specified with the OPTIONAL clause in the FILE-CONTROL paragraph of the Environment Division and is not present, the standard end of file processing is not performed.

5. If a CLOSE statement without the REEL or UNIT option has been executed for a file, a READ, WRITE, SEEK or SUSPEND statement for that file must not be executed unless an intervening OPEN statement for that file is executed.

CLOSE OPTION	File Type				
	Non-Reel	Sequential Single-reel/unit	Sequential Multi-reel/unit	Random Single-unit	Random Multi-unit
CLOSE	C	C,G	C,G,A	C	C
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A	C,E	C,E
CLOSE WITH NO REWIND	X	C,B	C,B,A	X	X
CLOSE REEL	X	X	F,G	X	X
CLOSE REEL WITH LOCK	X	X	F,D	X	X
CLOSE REEL WITH NO REWIND	X	X	F,B	X	X
CLOSE UNIT	X	X	F	X	X
CLOSE UNIT WITH LOCK	X	X	F,D	X	X

Figure 7-5. Relationship of Types of Files and the Options of the CLOSE Statement

7.13 THE COMPUTE STATEMENT

7.13.1 FUNCTION

The COMPUTE statement assigns to one or more data items the value of a numeric data item, literal or arithmetic expression.

7.13.2 GENERAL FORMAT

```
COMPUTE identifier-1 [ ROUNDED] [, identifier-2 [ROUNDED] ] ...

( FROM  )   ( identifier-n          )
{  =    }   { literal-1             } [; ON SIZE ERROR imperative-statement]
( EQUALS)   ( arithmetic-expression )
```

7.13.3 SYNTAX RULES

1. Literal-1 must be a numeric literal.

2. Each identifier must refer to an elementary numeric item, except that identifiers that appear only to the left of

```
( FROM  )
{  =    }   may describe data items that contain editing symbols.
( EQUALS)
```

3. The arithmetic expression option permits the use of any meaningful combination of identifiers, numeric literals, and arithmetic operators, parenthesized as required. (See 7.3, Arithmetic Expressions).

4. The maximum size of each operand is eighteen decimal digits.

7.13.4 GENERAL RULES

1. See 7.6.1, The ROUNDED Option; 7.6.2, The SIZE ERROR Option; 7.6.6, Multiple Results in Arithmetic Statements; and 3.2.1.2.5, Special Registers.

2. The identifier-n and literal-1 options provide a method for setting the values of identifier-1, identifier-2, etc., equal to the value of identifier-n or literal-1.

3. The words FROM and EQUALS are equivalent to each other and to the symbol '=' They may be used interchangeably and the choice is generally made for readability.

4. If more than one identifier is specified for the result of the operation, that is preceding FROM, =, or EQUALS, the value of the arithmetic expression is computed, and then this value or the value of literal-1, or identifier-n is stored as the new value of each of identifier-1, identifier-2, etc., in turn.

5. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY and DIVIDE.

   Thus, each implementor will indicate the techniques used in handling arithmetic expressions.

7.14 THE COPY STATEMENT

7.14.1 FUNCTION

The COPY statement incorporates procedures from a library into the source program.

7.14.2 GENERAL FORMAT

$$\left\{\begin{array}{l}\text{paragraph-name.}\\ \text{section-name } \underline{\text{SECTION}}\ [\,\text{priority-number}\,]\ .\end{array}\right\}\quad \underline{\text{COPY}}\ \text{library-name}$$

$$\left[\underline{\text{REPLACING}}\quad \left\{\begin{array}{l}\text{word-1}\\ \text{identifier-1}\end{array}\right\}\quad \underline{\text{BY}}\quad \left\{\begin{array}{l}\text{word-2}\\ \text{identifier-2}\end{array}\right\}\quad \left[,\ \left\{\begin{array}{l}\text{word-3}\\ \text{identifier-3}\end{array}\right\}\quad \underline{\text{BY}}\quad \left\{\begin{array}{l}\text{word-4}\\ \text{identifier-4}\end{array}\right\}\right]\ldots\right].$$

7.14.3 GENERAL RULES

1. For a discussion of the COPY function see Chapter 9, The COBOL Library.

7.15 THE DISPLAY STATEMENT

7.15.1 FUNCTION

The DISPLAY statement causes low volume data to be transferred to an appropriate hardware device.

7.15.2 GENERAL FORMAT

```
DISPLAY   {literal-1   }   [, literal-2   ]  ...  [UPON mnemonic-name]
          {identifier-1}   [, identifier-2]
```

7.15.3 SYNTAX RULES

1. The mnemonic-name is associated with a hardware device in the SPECIAL-NAMES paragraph in the Environment Division.

2. Each literal may be any figurative constant, except ALL.

7.15.4 GENERAL RULES

1. The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed.

2. The implementor will define, for each hardware device, the size of a data transfer.

3. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

4. If the hardware device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred.

5. If a hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:

   a. If the size of the data item being transferred exceeds the size of the data that the hardware device is capable of receiving in a single transfer, the data beginning with the left most character is stored aligned to the left in the receiving hardware device, and additional data is requested.

   b. If the size of the data item that the hardware device is capable of receiving exceeds the size of the data item being transferred, the transferred data is stored aligned to the left in the receiving hardware device.

6. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered.

7. If the UPON phrase is not specified, the implementor's standard display device is used.

7.16 THE DIVIDE STATEMENT

7.16.1 FUNCTION

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

7.16.2 GENERAL FORMAT

Format 1

```
DIVIDE { identifier-1 } INTO identifier-2 [ROUNDED] [, identifier-3 [ROUNDED]] ...
       { literal-1    }

      [; ON SIZE ERROR imperative-statement]
```

Format 2

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }      { literal-2    }

      [, identifier-4 [ROUNDED]] ... [; ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }    { literal-2    }

      [, identifier-4 [ROUNDED]] ... [; ON SIZE ERROR imperative-statement]
```

Format 4

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }      { literal-2    }

         REMAINDER identifier-4      [; ON SIZE ERROR imperative-statement]
```

Format 5

```
DIVIDE { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }    { literal-2    }

         REMAINDER identifier-4      [; ON SIZE ERROR imperative-statement]
```

7.16.3 SYNTAX RULES

1. Each identifier must refer to a numeric elementary item, except, in Formats 2 and 3, where any identifiers that appear only to the right of the word GIVING may refer to data items that contain editing symbols.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is the data item resulting from the superimposition of all receiving data items aligned on their decimal points, must not contain more than eighteen digits.

7.16.4 GENERAL RULES

1. See 7.6.1, The ROUNDED Option; 7.6.2, The SIZE ERROR Option; 7.6.6, Multiple Results in Arithmetic Statements; and 3.2.1.2.5, Special Registers for a description of these functions.

2. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient; similarly for identifier-1 or literal-1 and identifier-3, etc.

3. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

4. When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

5. Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. A remainder in COBOL is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If the ROUNDED option is specified, the quotient is rounded after the remainder is determined.

7.17 THE ENTER STATEMENT

7.17.1 FUNCTION

The ENTER statement provides a means of allowing the use of more than one language in the same program.

7.17.2 GENERAL FORMAT

ENTER language-name [routine-name]

7.17.3 SYNTAX RULES

1. The language-name may refer to any programming language which the implementor specifies may be entered through COBOL. Language-name is specified by the implementor.

2. If the statements in the entered language cannot be written in-line, a routine-name is given to identify the portion of the other-language coding to be executed at this point in the procedure sequence. A routine-name is a COBOL word and it may be referred to only in an ENTER sentence.

3. If the other-language statements can be written in-line, the routine-name option is not used. The sentence ENTER COBOL must follow the last other-language statement in order to indicate to the compiler where a return to COBOL source language takes place.

7.17.4 GENERAL RULES

1. The other-language statements are executed in the object program as if they had been compiled into the object program following the ENTER statement.

2. Implementors will specify, for their compilers, all details on how the other language(s) are to be written.

7.18 THE EXAMINE STATEMENT

7.18.1 FUNCTION

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

7.18.2 GENERAL FORMAT

```
                    ⎧TALLYING   ⎧UNTIL FIRST⎫  ⎧literal-1   ⎫ [REPLACING BY ⎧literal-2   ⎫]⎫
                    ⎪           ⎨ALL        ⎬  ⎩identifier-1⎭               ⎩identifier-2⎭ ⎪
                    ⎪           ⎩LEADING    ⎭                                              ⎪
EXAMINE identifier  ⎨                                                                      ⎬
                    ⎪           ⎧ALL        ⎫                                              ⎪
                    ⎪REPLACING  ⎨LEADING    ⎬  ⎧literal-3   ⎫       BY      ⎧literal-4   ⎫ ⎪
                    ⎩           ⎩[UNTIL] FIRST⎭⎩identifier-3⎭               ⎩identifier-4⎭ ⎭
```

7.18.3 SYNTAX RULES

1. The description of the identifier must be such that usage is display (explicitly or implicitly).

2. Each identifier-n must name a single character data item belonging to a class consistent with that of identifier. Each literal must consist of a single character belonging to a class consistent with that of identifier; in addition, each literal may be any figurative constant, except ALL.

3. A signed numeric literal is not permitted in the EXAMINE statement.

7.18.4 GENERAL RULES

1. Examination proceeds as follows:

   a. For nonnumeric data items, examination starts at the left-most character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.

   b. If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may possess an operational sign. Examination starts at the left-most character and proceeds to the right. Each character is examined in turn. If the letter 'S' is used in the PICTURE character-string of the data item description to indicate the presence of an operational sign, the sign is completely ignored by the EXAMINE statement.

2. The TALLYING option creates an integral count which replaces the value of a special register called TALLY (see 3.2.1.2.5.1, TALLY). The count represents the number of:

   a. Occurrences of literal-1 or identifier-1 when the ALL option is used.

   b. Occurrences of literal-1 or identifier-1 prior to encountering a character other than literal-1 or identifier-1 when the LEADING option is used.

   c. Characters not equal to literal-1 or identifier-1 encountered before the first occurrence of literal-1 or identifier-1 when the UNTIL FIRST option is used.

3. When either of the REPLACING options is used, the replacement rules are as follows, subject to General Rule 2:

   a. When the ALL option is used, then literal-2, identifier-2 or literal-4, identifier-4 is substituted for each occurrence of literal-1, identifier-1 or literal-3, identifier-3.

   b. When the LEADING option is used, the substitution of literal-2, identifier-2 or literal-4, identifier-4 terminates as soon as a character other than literal-1, identifier-1 or literal-3, identifier-3 or the right-hand boundary of the data item is encountered.

   c. When the UNTIL FIRST option is used, the substitution of literal-2 identifier-2 or literal-4, identifier-4 terminates as soon as literal-1, identifier-1 or literal-3, identifier-3 or the right-hand boundary of the data item is encountered.

   d. When the FIRST option is used, the first occurrence of literal-1, identifier-1 or literal-3, identifier-3 is replaced by literal-2, identifier-2 or literal-4, identifier-4.

7.19 THE EXIT STATEMENT

7.19.1 FUNCTION

The EXIT statement provides a common end point for a series of procedures, or marks the logical end of a called program.

7.19.2 GENERAL FORMAT

EXIT [PROGRAM].

7.19.3 SYNTAX RULES

1. The EXIT statement must appear in a sentence by itself.

2. The EXIT sentence must be preceded by a paragraph-name and be the only sentence in the paragraph.

7.19.4 GENERAL RULES

1. It is sometimes necessary to transfer control to the end point of a series of procedures. This is normally done by transferring control to the next paragraph or section, but in some cases this does not have the required effect. For instance, the point to which control is to be transferred may be at the end of a range of procedures governed by a PERFORM or at the end of a declarative section. The EXIT statement is provided to enable a procedure-name to be associated with such a point.

2. If control reaches an EXIT statement without the PROGRAM option and no associated PERFORM or USE statement is active or if control reaches an EXIT PROGRAM statement and no CALL statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

3. If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

7.20 THE GENERATE STATEMENT

7.20.1 FUNCTION

The GENERATE statement links the Procedure Division to the Report Writer (described in the Report Section of the Data Division) at process time.

7.20.2 GENERAL FORMAT

GENERATE identifier

7.20.3 SYNTAX RULES

1. Identifier represents a TYPE DETAIL report group or an RD entry.

7.20.4 GENERAL RULES

1. If identifier represents the name of a TYPE DETAIL report group, the GENERATE statement does all the automatic operations within a Report Writer and produces an actual output DETAIL report group, at process time, on the output medium. This is called detail reporting.

2. If identifier represents the name of a RD entry, the GENERATE statement does all the automatic operations of the Report Writer and updates the FOOTING report group(s) within a particular report description without producing an actual DETAIL report group associated with the report. In this case, all SUM counters associated with the report description are algebraically incremented. This is called summary reporting. If more than one TYPE DETAIL report group is specified, all SUM counters are algebraically incremented each time a GENERATE statement is executed.

3. A GENERATE statement, implicitly in both detail and summary reporting, produces the following automatic operations (if defined):

   a. Steps and tests the LINE-COUNTER and/or PAGE-COUNTER to produce appropriate PAGE or OVERFLOW FOOTING and/or PAGE or OVERFLOW HEADING report groups.

   b. Recognizes any specified CONTROL breaks to produce appropriate CONTROL FOOTING and/or CONTROL HEADING report groups.

c. Accumulates into the SUM counters all specified identifier(s). Resets the SUM counters on an associated control break. Performs an updating procedure between control break levels for each set of SUM counters.

d. Executes any specified routines defined by a USE statement before generation of the associated report group(s).

4. During the execution of the first GENERATE statement, the following report groups associated with the report, if specified, are produced in the order:

   a. REPORT HEADING report group.

   b. PAGE HEADING report group.

   c. ALL CONTROL HEADING report groups in the order FINAL, major to minor.

   d. The DETAIL report group, if specified in the GENERATE statement.

5. If a control break is recognized at the time of execution of a GENERATE statement (other than the first that is executed for a report), all CONTROL FOOTING report groups specified for the report are produced from the minor report group up to and including the report group specified for the identifier which caused the control break. Then, the CONTROL HEADING report group(s) specified for the report, from the report group specified for the identifier that caused the control break down to the minor report group, are produced in that order. The DETAIL report group specified in the GENERATE statement is then produced.

6. Data is moved to the data item in the Report Group Description entry of the Report Section, and is edited under control of the Report Writer according to the same rules for movement and editing as described for the MOVE statement.

7.21 THE GO TO STATEMENT

7.21.1 FUNCTION

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

7.21.2 GENERAL FORMAT

Format 1

GO TO [procedure-name-1]

Format 2

GO TO procedure-name-1 [, procedure-name-2] ..., procedure-name-n

DEPENDING ON identifier

7.21.3 SYNTAX RULES

1. Each procedure name is the name of a paragraph or section in the Procedure Division of the program.

2. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

7.21.4 GENERAL RULES

1. When a GO TO statement represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure name if the GO TO statement has been modified by an ALTER statement.

2. If procedure-name-1 is not specified in Format 1, an ALTER statement, referring to this GO TO statement, must be executed prior to the execution of this GO TO statement.

3. When, in Format 1, the GO TO statement is referred to by an ALTER statement, the following rules apply regardless of whether or not procedure-name-1 is specified.

   a. The GO TO statement must have a paragraph-name.

   b. The GO TO statement must be the only statement in the paragraph.

4. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1, procedure-name-2, etc., depending on the value of the identifier being 1, 2, ..., n. If the value of identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

5. If a GO TO statement represented by Format 1 appears in an imperative sentence, it must appear as the only statement or the last statement in a sequence of imperative statements.

7.22 THE HOLD STATEMENT

7.22.1 FUNCTION

The HOLD statement provides, in an asynchronous environment, a delay point that causes synchronous processing to be resumed.

7.22.2 GENERAL FORMAT

$$\underline{\text{HOLD}} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \text{section-name-1} \left[\text{, section-name-2} \right] \ldots \end{array} \right\}$$

7.22.3 SYNTAX RULES

1. The section-names used as operands of the HOLD statement must be names of sections defined in a USE FOR RANDOM PROCESSING Section in a USE Declarative.

7.22.4 GENERAL RULES

1. A HOLD ALL statement may only be used in in-line procedures. The statement ensures that all previously initiated asynchronous procedures have been completed before any statements following the HOLD ALL statement are executed.

2. For in-line procedural statements, when one or more sections are named as operands of the HOLD statement, the HOLD statement ensures that all previously initiated asynchronous procedures pertaining to the sections named have been completed before any statements following the HOLD statements are executed.

3. For out-of-line procedural statements, the operand of the HOLD statement must name the section in which the HOLD statement appears. The execution of a HOLD statement forces procedural statements following the HOLD statement, within a particular processing cycle, to be processed in the order in which the out-of-line processing cycles were initiated. Asynchronous processing must not be reinitiated in the out-of-line set of procedures following the HOLD statement. When a HOLD statement is not specified in the out-of-line procedures, the out-of-line processing cycles are processed and completed in an asynchronous manner regardless of the order in which the cycles were initiated.

4. A HOLD statement is meaningful only when used with asynchronous processing cycles initiated by a PROCESS statement.

7.23 THE IF STATEMENT

7.23.1 FUNCTION

The IF statement causes a condition (see 7.4, Conditions) to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

7.23.2 GENERAL FORMAT

$$\underline{\text{IF}}\ \text{condition}\quad ;\quad \left\{\begin{array}{l}\text{statement-1}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}$$

$$\left\{\begin{array}{ll}\text{; }\underline{\text{ELSE}} & \text{statement-2}\\ \text{; }\underline{\text{ELSE}} & \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}$$

7.23.3 SYNTAX RULES

1. Statement-1 and statement-2 represent either a conditional statement or an imperative statement, and either may be followed by a conditional statement.
2. The phrase 'ELSE NEXT SENTENCE' may be omitted if it immediately precedes the terminal period of the sentence.

7.23.4 GENERAL RULES

1. When an IF statement is executed, the following action is taken:
   a. If the condition is true, the statements immediately following the condition (represented by statement-1) are executed and control then passes implicitly to the next sentence.
   b. If the condition is false, either the statements following ELSE are executed or, if the ELSE clause is omitted, the next sentence is executed.
2. When an IF statement is executed and the NEXT SENTENCE phrase is present, control passes explicitly to the next sentence depending on the truth value of the condition and the placement of the NEXT SENTENCE phrase in the statement.
3. Statement-1 and statement-2 may contain an IF statement. In this case, the IF statement is said to be nested.

IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

4. When control is transferred to the next sentence, either implicitly or explicitly, control passes to the next sentence as written or to a return mechanism of a PERFORM or a USE statement.

7.24 THE INITIATE STATEMENT

7.24.1 FUNCTION

The INITIATE statement begins processing of a report.

7.24.2 GENERAL FORMAT

```
INITIATE     { report-name-1      [, report-name-2]  ... }
             { ALL                                      }
```

7.24.3 SYNTAX RULES

1. Each report-name must be defined by a Report Description entry in the Report Section of the Data Division.

7.24.4 GENERAL RULES

1. The INITIATE statement resets all data-name entries that contain SUM clauses associated with this report; the Report Writer controls for all the TYPE report groups that are associated with this report are set up in their respective order.

2. The PAGE-COUNTER register, if specified, is set to one (1) prior to or during the execution of the INITIATE statement. If a different starting value for the associated PAGE-COUNTER other than one (1) is desired, the programmer may reset the counter following the INITIATE statement.

3. The LINE-COUNTER register, if specified, is set to zero prior to or during the execution of the INITIATE statement.

4. If ALL is specified, all report-name(s) defined by RD entries are initiated.

5. The INITIATE statement does not open the file with which the report is associated, however the associated file must be open at the time the INITIATE statement is executed. The INITIATE statement performs Report Writer functions for individually described report programs analogous to the input-output functions that the OPEN statement performs for individually described files.

6. A second INITIATE for a particular report-name may not be executed unless a TERMINATE statement has been executed for that report-name subsequent to the first INITIATE statement.

7.25 THE MOVE STATEMENT

7.25.1 FUNCTION

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

7.25.2 GENERAL FORMAT

$$\underline{\text{MOVE}}\ \left\{\begin{matrix}\left[\begin{matrix}\underline{\text{CORR}}\\ \underline{\text{CORRESPONDING}}\end{matrix}\right]\ \text{identifier-1}\\ \text{literal}\end{matrix}\right\}\ \ \underline{\text{TO}}\ \ \text{identifier-2}\ \ \left[,\ \text{identifier-3}\right]\ \ldots$$

7.25.3 SYNTAX RULES

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3, represent the receiving area.

2. CORR is an abbreviation for CORRESPONDING.

7.25.4 GENERAL RULES

1. If the CORRESPONDING phrase is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given in 7.6.3, The CORRESPONDING Option. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

2. The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, etc. The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, etc., is evaluated immediately before the data is moved to the respective data item.

3. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in 6.31, The PICTURE Clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO (ZEROS, ZEROES) belongs to the category numeric. The figurative constant SPACE (SPACES) belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

   The following rules apply to an elementary move between these categories:

   a. A numeric edited, alphanumeric edited, the figurative constant SPACE, or an alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

   c. A numeric literal, or a numeric data item whose implicit decimal point is not immediately to the right of the least significant digit, must not be moved to an alphanumeric or alphanumeric edited data item.

   d. All other elementary moves are legal and are performed according to the rules given in General Rule 4.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

   a. When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item, justification and any necessary space-filling takes place as defined under the JUSTIFIED clause. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled.

   b. When a numeric or numeric edited item is a receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the JUSTIFIED clause, except where zeros are replaced because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, the excess digits are truncated. When a data item described as alphanumeric is the sending item, it is moved as though it was described as an unsigned numeric integer item. If the sending item contains any nonnumeric characters, the results are undefined.

c. When a receiving field is described as alphabetic and the sending data item contains any nonalphabetic characters, the results are undefined.

5. An index data item cannot appear as an operand of a MOVE statement.

6. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area.

7. Data in the following chart represents the legality of the Move/General Rule Reference. The general rule reference indicates the rule that prohibits the move or the behavior of a legal move.

Category of Sending Data Item		Category of Receiving Data Item		
		Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Numeric Non-integer Numeric Edited
ALPHABETIC		Yes/4a	Yes/4a	No/3a
ALPHANUMERIC		Yes/4c	Yes/4a	Yes/4b
ALPHANUMERIC EDITED		Yes/4c	Yes/4a	No/3a
NUMERIC	INTEGER	No/3b	Yes/4a	Yes/4b
	NON-INTEGER	No/3b	No/3c	Yes/4b
NUMERIC EDITED		No/3b	Yes/4a	No/3a

7.26 THE MULTIPLY STATEMENT

7.26.1 FUNCTION

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

7.26.2 GENERAL FORMAT

Format 1

```
MULTIPLY  { identifier-1 }  BY identifier-2  [ROUNDED] [, identifier-3 [ROUNDED]]...
          { literal-1    }

          [; ON SIZE ERROR imperative-statement]
```

Format 2

```
MULTIPLY  { identifier-1 }  BY  { identifier-2 }  GIVING identifier-3 [ROUNDED]
          { literal-1    }      { literal-2    }

          [, identifier-4 [ROUNDED] ] ...

          [; ON SIZE ERROR imperative-statement]
```

7.26.3 SYNTAX RULES

1. Each identifier must refer to a numeric elementary item, except in Format 2, where any identifiers that appear only to the right of the word GIVING may refer to data items that contain editing symbols.

2. Each literal must be a numeric literal.

3. The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is that data item resulting from the superimposition of all receiving data items aligned on their decimal points, must not contain more than eighteen (18) digits.

7.26.4 GENERAL RULES

1. See 7.6.1, The ROUNDED Option 7.6.2, The SIZE ERROR Option; 7.6.6, Multiple Results in Arithmetic Statements; and 3.2.1.2.5, Special Registers.

2. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product; similarly for identifier-1 or literal-1 and identifier-3, etc.

3. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

7.27 THE OPEN STATEMENT

7.27.1 FUNCTION

The OPEN statement initiates the processing of files. It performs checking and/or writing of labels and other input-output operations.

7.27.2 GENERAL FORMAT

```
      ⎧ INPUT file-name-1 ⎡REVERSED      ⎤ ⎡, file-name-2 ⎡REVERSED      ⎤⎤...  ⎫
      ⎪                   ⎣WITH NO REWIND⎦ ⎣              ⎣WITH NO REWIND⎦⎦     ⎪
OPEN  ⎨ OUTPUT file-name-3 [WITH NO REWIND] [, file-name-4 [WITH NO REWIND]]... ⎬...
      ⎪                                                                         ⎪
      ⎩ I-O file-name-5 [, file-name-6] ...                                     ⎭
```

7.27.3 SYNTAX RULES

1. The I-O phrase pertains only to mass storage files.

2. The REVERSED and WITH NO REWIND phrases do not apply to mass storage processing.

7.27.4 GENERAL RULES

1. The OPEN statement must not be applied to sort-files, but must be applied to all other files. The OPEN statement for a file must be executed prior to the first READ, WRITE, SEEK or SUSPEND statement for that file.

2. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

3. The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record.

4. If a label record is specified for the file, the label is processed according to the implementor's standard beginning label convention. The behavior of the OPEN statement when a label record is specified but not present, or when a label record is not specified but is present, is undefined. If specified by the USE statement, a user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. If label records are indicated as present by a LABEL RECORDS clause the user's beginning label procedure, if specified by a USE statement, is executed before or after (as indicated) checking but subsequent to writing the first label.

5. The REVERSED and the NO REWIND phrases can only be used with sequential single reel/unit (see 7.12, The CLOSE Statement).

6. If the external medium for the file permits rewinding, the following rules apply:

   a. When neither the REVERSED nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

   b. When either the REVERSED or the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned. When the REVERSED phrase is specified, the file must be positioned at its end. When the NO REWIND phrase is specified, the file must be positioned at its beginning.

7. When the REVERSED phrase is specified, the subsequent READ statements for the file make the data records of the file available in reverse order; that is, starting with the last record.

8. If an input file is designated with the OPTIONAL clause in the FILE-CONTROL paragraph of the Environment Division, the object program causes an interrogation for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the imperative statement in the AT END phrase to be executed.

9. The I-O option permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being initially created.

10. When I-O is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. The label is checked in accordance with the implementor's specified conventions for input-output label checking.

    b. The user's beginning label procedure, if one is specified by the USE statement, is executed according to paragraph 7.41.4, General Rule 1.

    c. The new label is written in accordance with the implementor's specified conventions for input-output label writing.

11. When processing mass storage files for which the access mode is sequential, the OPEN statement supplies the initial address of the first record to be accessed.

12. The contents of the data-names specified in the FILE-LIMIT clause of the FILE-CONTROL paragraph are checked by the Mass Storage Control System only when the OPEN statement is executed. The FILE-LIMIT clause is dynamic only to this extent.

7.28 THE PERFORM STATEMENT

7.28.1 FUNCTION

The PERFORM statement is used to depart from the normal sequence of procedures in order to execute one or more procedures either a specified number of times or until a specified condition is satisfied and to provide a means of return to the normal sequence.

7.28.2 GENERAL FORMAT

Format 1

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\ \text{procedure-name-2}\right]$$

Format 2

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\ \text{procedure-name-2}\right]$$

$$\left\{\begin{matrix}\text{identifier-1}\\ \text{integer-1}\end{matrix}\right\}\ \underline{\text{TIMES}}$$

Format 3

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\ \text{procedure-name-2}\right]\ \underline{\text{UNTIL}}\ \text{condition-1}$$

Format 4

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\ \text{procedure-name-2}\right]$$

$$\underline{\text{VARYING}}\ \left\{\begin{matrix}\text{identifier-1}\\ \text{index-name-1}\end{matrix}\right\}\ \underline{\text{FROM}}\ \left\{\begin{matrix}\text{identifier-2}\\ \text{index-name-2}\\ \text{literal-1}\end{matrix}\right\}$$

$$\underline{\text{BY}}\ \left\{\begin{matrix}\text{identifier-3}\\ \\ \text{literal-2}\end{matrix}\right\}\ \underline{\text{UNTIL}}\ \text{condition-1}$$

$$\left[\ \underline{\text{AFTER}}\ \left\{\begin{matrix}\text{identifier-4}\\ \text{index-name-3}\end{matrix}\right\}\ \underline{\text{FROM}}\ \left\{\begin{matrix}\text{identifier-5}\\ \text{index-name-4}\\ \text{literal-3}\end{matrix}\right\}\right.$$

$$\underline{\text{BY}}\ \left\{\begin{matrix}\text{identifier-6}\\ \\ \text{literal-4}\end{matrix}\right\}\ \underline{\text{UNTIL}}\ \text{condition-2}$$

Format 4 (Contd.)

$$\left[\ \underline{\text{AFTER}}\quad \left\{\begin{array}{l}\text{identifier-7}\\ \text{index-name-5}\end{array}\right\}\right.$$

$$\left.\quad \underline{\text{FROM}}\quad \left\{\begin{array}{l}\text{identifier-8}\\ \text{index-name-6}\\ \text{literal-5}\end{array}\right\}\quad \underline{\text{BY}}\quad \left\{\begin{array}{l}\text{identifier-9}\\ \\ \text{literal-6}\end{array}\right\}\quad \underline{\text{UNTIL}}\ \text{condition-3}\ \right]\ \Bigg]$$

7.28.3 SYNTAX RULES

1. Each procedure-name is the name of a section or paragraph in the Procedure Division.

2. Each identifier represents a numeric elementary item described in the Data Division. In Format 2 and Format 4 with the optional phrase AFTER, each identifier represents a numeric item with no positions to the right of the assumed decimal point.

3. Each literal represents a numeric literal.

4. The words THRU and THROUGH are equivalent.

7.28.4 GENERAL RULES

1. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows:

   a. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return is after the last statement of procedure-name-1.

   b. If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

   c. If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

   d. If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

2. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

3. If control passes to these procedures by means other than a PERFORM statement, control will pass through the last statement of the procedure to the following statement as if no PERFORM statement mentioned these procedures.

4. The PERFORM statements operate as follows with Rule 3 above applying to all formats:

   a. Format 1 is the basic PERFORM statement. A procedure referred to by this type of PERFORM statement is executed once and then control passes to the statement following the PERFORM statement.

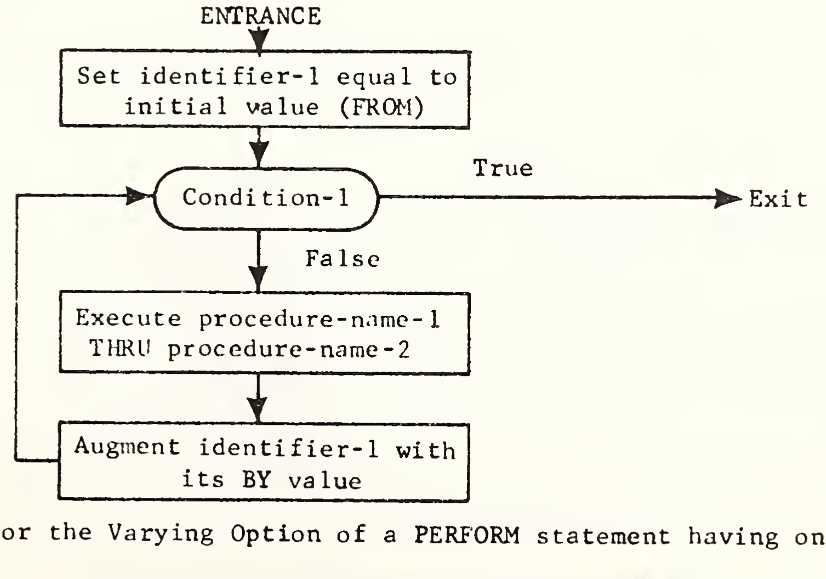
   b. Format 2 is the TIMES option. When the TIMES option is used, the procedures are performed the number of times specified by the initial value of identifier-1 or integer-1 for that execution. When the PERFORM statement is executed, the value of integer-1 must be positive. If the initial value of identifier-1 is negative or zero, control passes immediately to the statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the statement following the PERFORM statement.

      During execution of the PERFORM statement, reference to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

c. Format 3 is the UNTIL option. The specified procedures are performed until the condition specified by the UNTIL phrase is true. The condition may be any condition as described in paragraph 7.4, Conditions. When the condition is true, control is transferred to the next statement after the PERFORM statement. If the condition is true when the PERFORM statement is entered, no transfer to procedure-name-1 takes place, and control is passed to the next statement following the PERFORM statement.

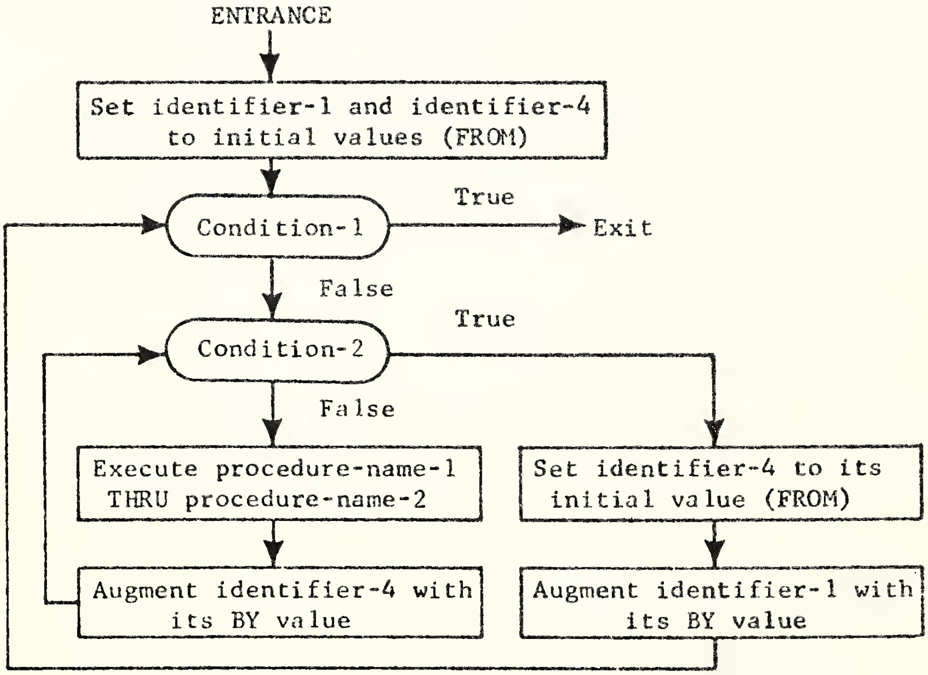
d. Format 4 is the VARYING option. This option is used to augment the value of one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING and FROM (starting value) phrases also refers to index-names. When index-names are used, the FROM and BY clauses have the same effect as in a SET statement.

In Format 4, when one identifier is varied, identifier-1 is set to its starting value (the value of identifier-2 or literal-1) at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL clause is false, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The value of identifier-1 is augmented by the specified increment or decrement value (the value of identifier-3 or literal-2) and condition-1 is evaluated again. The cycle continues until this expression is true; at which point, control passes to the statement following the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control passes directly to the statement following the PERFORM statement.


ENTRANCE
Set identifier-1 equal to initial value (FROM)
Condition-1
True
Exit
False
Execute procedure-name-1 THRU procedure-name-2
Augment identifier-1 with its BY value

Flow Chart for the Varying Option of a PERFORM statement having one condition.

In Format 4, when two identifiers are varied, identifier-1 and identifier-4 are set to their initial values (the values of identifier-2 and identifier-5, respectively). During execution, these initial values must be positive. After initializing the identifiers, condition-1 is evaluated; if true, control is passed to the statement following the PERFORM statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, then identifier-4 is augmented by identifier-6 or literal-4 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When condition-2 is true, identifier-4 is set to its initial value (the value of identifier-5 or literal-3), identifier-1 is augmented by identifier-3 and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true. Identifier-3 and identifier-6 must not be zero. During execution of the PERFORM statement, reference to index names or identifiers of the FROM clause has no effect in altering the number of times the procedures are to be executed. Changing a value of index-names or identifiers of the VARYING clause or identifiers of the BY clause, however, will change the number of times procedures are executed.


ENTRANCE
Set identifier-1 and identifier-4 to initial values (FROM)
Condition-1
True
Exit
False
Condition-2
True
False
Execute procedure-name-1 THRU procedure-name-2
Augment identifier-4 with its BY value
Set identifier-4 to its initial value (FROM)
Augment identifier-1 with its BY value

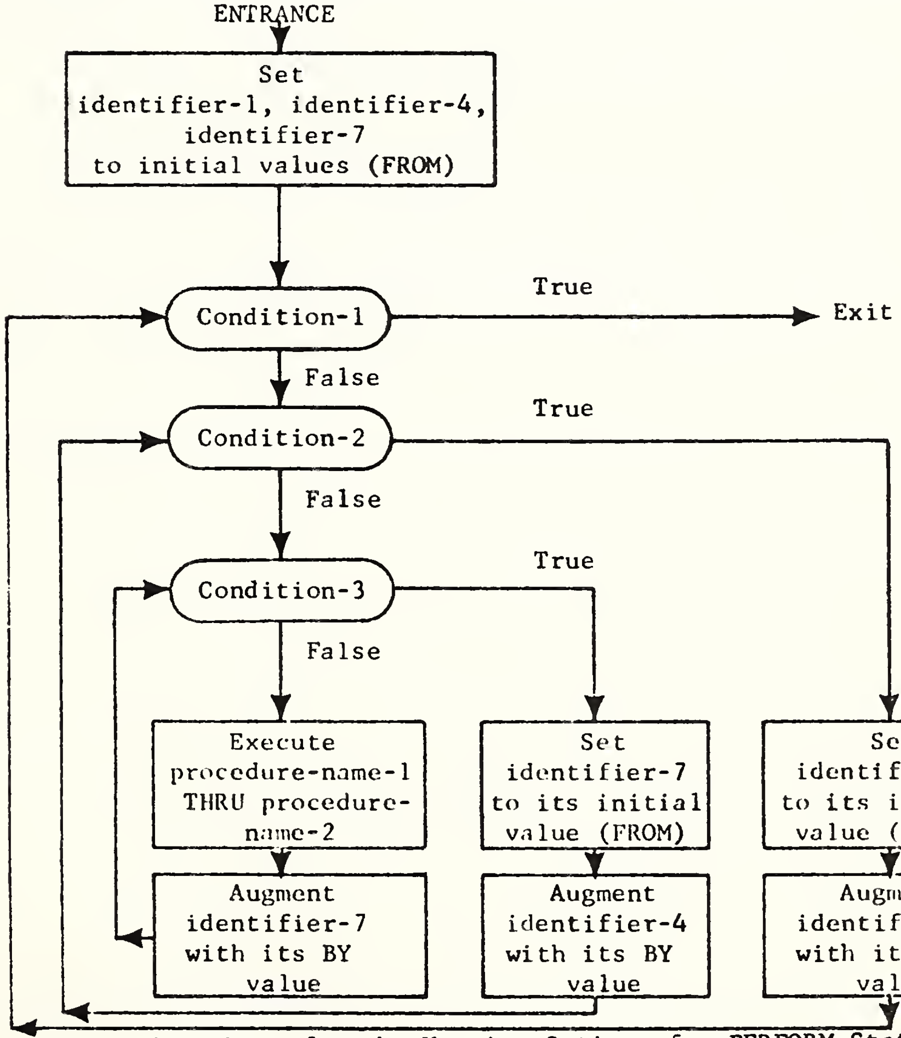
Flow Chart for the Varying Option of a Perform Statement having two conditions (Format 4).

At the termination of the PERFORM statement identifier-4 contains its initial value, while identifier-1 has a value that exceeds the last used setting by an increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-1 and identifier-4 contain their initial values

When two identifiers are varied, identifier-4 goes through a complete cycle (FROM, BY, UNTIL) each time identifier-1 is varied.

For three identifiers the mechanism is the same as for two identifiers except that identifier-7 goes through a complete cycle each time that identifier-4 is augmented by identifier-6 or literal-4, which in turn goes through a complete cycle each time identifier-1 is varied.

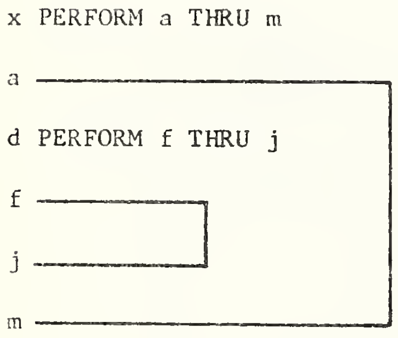
The following flow chart illustrates the logic of the PERFORM statement when three identifiers are varied.

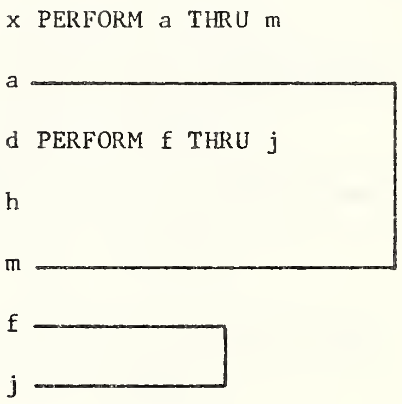

ENTRANCE
Set identifier-1, identifier-4, identifier-7 to initial values (FROM)
Condition-1
True
Exit
False
Condition-2
True
False
Condition-3
True
False
Execute procedure-name-1 THRU procedure-name-2
Set identifier-7 to its initial value (FROM)
Set identifier-4 to its initial value (FROM)
Augment identifier-7 with its BY value
Augment identifier-4 with its BY value
Augment identifier-1 with its BY value

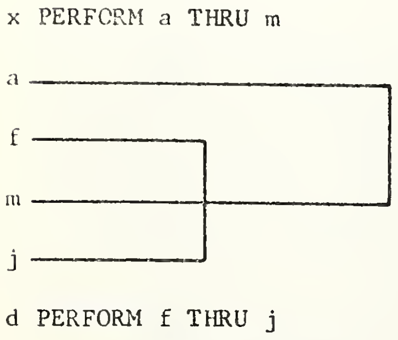
Flow Chart for the Varying Option of a PERFORM Statement Having Three Conditions

After the completion of a Format 4 PERFORM statement, identifier-4 and identifier-7 contain their initial values, while identifier-1 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-1, identifier-4 and identifier-7 all contain their initial values.

5. If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement. must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See the illustrations below:







6. A PERFORM statement that appears in a section whose priority-number is less than the segment limit, can have within its range only the following:

   a. Sections each of which has a priority-number less than 50, or

   b. Sections wholly contained in a single segment whose priority-number is greater than 49.

7. A PERFORM statement that appears in a section whose priority-number is equal to or greater than the segment limit, can have within its range only the following:

   a. Sections each of which has the same priority-number as that containing the PERFORM statement, or

   b. Sections with a priority-number that is less than the segment limit.

8. When a procedure-name in a segment with a priority-number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority-number, the segment referred to is made available in its initial state for each execution of the PERFORM statement.

7.29 THE PROCESS STATEMENT

7.29.1 FUNCTION

The PROCESS statement initiates a set of out-of-line procedures under the control of an Asynchronous Control System.

7.29.2 GENERAL FORMAT

$$\underline{\text{PROCESS}}\ \text{section-name}\ \left[\underline{\text{FROM}}\ \ \text{identifier}\right]\left[\ \underline{\text{USING}}\ \begin{Bmatrix}\text{area-name}\\ \text{record-name}\end{Bmatrix}\ \right]$$

7.29.3 SYNTAX RULES

1. Section-name must be the name of an out-of-line procedure defined in a USE FOR RANDOM PROCESSING section in the Declarative Section. A PROCESS statement cannot appear in a USE FOR RANDOM PROCESSING section.

2. Area-name, when used, must be the name of a Saved Area Description entry (SA) defined in the File Section of the Data Division.

3. Record-name, when used, must be the name of a level-number 01 record description entry, which is subordinate to an SA entry.

4. The USING phrase must be used when more than one Saved Area Description entry is specified.

7.29.4 GENERAL RULES

1. Data that is to be processed by an out-of-line procedure must be placed in a Saved Area either by in-line procedural statements which are executed prior to the execution of the PROCESS statement in the in-line procedural statements, or by using the optional phrase FROM in the PROCESS statement. In addition, any data working-storage area that is required for the execution of a cycle of an out-of-line procedure should be specified as a part of the associated Saved Area Description entry.

2. One Saved Area record is automatically associated at object time with each out-of-line processing cycle. No more than one processing cycle has access to a single Saved Area record at any one time. The specific Saved Area record associated with an out-of-line processing cycle is released for further storage assignment by the Asynchronous Control System upon completion of that processing cycle.

3. The PROCESS statement is meaningful only when used in conjunction with an Asynchronous Control System (ACS).

4. When the optional phrase FROM is used, moving of identifier to area-name or record-name takes place in accordance with the rules specified for the MOVE statement without the CORRESPONDING option.

5. The processing of data in the Saved Area by out-of-line procedural statements may be performed asynchronously. Therefore, in-line procedures must not refer to any data being processed in the out-of-line set of procedures. Conversely, the out-of-line set of procedures must not refer to any data being processed in the in-line set of procedures.

7.30 THE READ STATEMENT

7.30.1 FUNCTION

1. For sequential file processing, the READ statement makes available the next logical record from an input file and allows performance of a specified imperative statement when end of file is detected.

2. For random file processing, the READ statement makes available a specified record from a mass storage file and allows performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item are found to be invalid.

7.30.2 GENERAL FORMAT

Format 1

READ file-name RECORD [INTO identifier] ; AT END imperative-statement

Format 2

READ file-name RECORD [INTO identifier] ; INVALID KEY imperative-statement

7.30.3 SYNTAX RULES

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record description. The storage area associated with identifier and the storage area which is the record area associated with the file-name must not be the same storage area. File-name must not represent a sort-file.

2. Format 1 is used only for non-mass storage files and for mass storage files in the sequential access mode.

3. Format 2 is used for mass storage files in the random access mode.

7.30.4 GENERAL RULES

1. An OPEN statement must be executed for a file prior to the execution of the first READ statement for that file.

2. If after reading the last logical record of a file another READ statement is initiated for that file, that last logical record is no longer available in its record area; the READ statement is then completed by the execution of the AT END phrase. After the AT END condition has been recognized for a file, a READ statement for that file must not be given without prior execution of a CLOSE statement and an OPEN statement for that file. The logical end of a file is specified in the FILE-LIMIT clause or the ASSIGN clause of the Environment Division.

3. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information that is present in the current record is accessible.

4. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

5. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

6. If a file described with the OPTIONAL clause is not present, the imperative statement in the AT END phrase is executed on the first READ. The standard end-of-file procedures are not performed (see the OPEN and USE statements, and the FILE-CONTROL paragraph in the Environment Division).

7. If the end of a tape reel or mass storage unit is recognized during execution of a READ statement the following operations are carried out:

   a. The standard ending reel/unit label procedure and the user's ending reel/unit label procedure, if specified, by the USE statement. The order of execution of these two procedures is specified by the USE statement.

   b. A tape/unit swap.

   c. The standard beginning reel/unit label procedure and the user's beginning reel/unit label procedure, if specified. The order of execution is again specified by the USE statement.

d. The first data record of the new reel/unit is made available.

For a mass storage file, this is only possible where the mass storage unit is in sequential access mode.

8. Format 2 is used for mass storage files in the random access mode. The READ statement implicitly performs the function of the SEEK statement for a specific mass storage file, unless a SEEK statement is executed for the specified record of this file prior to the execution of the READ statement for that specified record. A SEEK statement is related to a subsequent READ statement only if both are in the in-line procedures or both are in the same processing cycle of an out-of-line procedure. If such files are accessed for a specified mass storage record and the contents of the associated ACTUAL KEY data item are invalid, the INVALID KEY phrase is executed.

9. If a mass storage file (either INPUT or I-O), in the random access mode is contained on more than one physical mass storage unit, and not all of the physical units are simultaneously available, the procedures for making the physical units available are specified by the implementor (see 5.4.1, The FILE-CONTROL Paragraph).

10. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available prior to the execution of any statement following the READ statement.

7.31 THE RELEASE STATEMENT

7.31.1 FUNCTION

The RELEASE statement transfers records to the initial phase of a SORT operation.

7.31.2 GENERAL FORMAT

RELEASE record-name [FROM identifier]

7.31.3 SYNTAX RULES

1. A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement for a file whose sort-file description contains a record-name.

2. Record-name must be the name of a logical record in the associated sort-file description and may be qualified.

3. Record-name and identifier must not refer to the same storage area.

7.31.4 GENERAL RULES

1. The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.

2. If the FROM phrase is used, the contents of the identifier data area are moved to record-name, then the contents of record-name are released to the sort-file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The information in the record area is no longer available, but the information in the data area associated with identifier is available.

3. After the RELEASE is executed, the logical record is no longer available. When control passes from the INPUT PROCEDURE, the file consists of all those records which were placed in it by the execution of RELEASE statements.

7.32 THE RETURN STATEMENT

7.32.1 FUNCTION

The RETURN statement obtains sorted records from the final phase of a SORT operation.

7.32.2 GENERAL FORMAT

RETURN file-name RECORD [INTO identifier] ; AT END imperative-statement

7.32.3 SYNTAX RULES

1. File-name must be described by a Sort File Description entry in the Data Division.

2. A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name.

3. The INTO phrase may only be used when the input file contains just one type of record. The storage area associated with identifier and the storage area which is the record area associated with file-name must not be the same storage area.

7.32.4 GENERAL RULES

1. When a file consists of more than one type of logical record, these records automatically share the storage area. This is equivalent to saying that there exists an implicit redefinition of the area, and only the information that is present in the current record is accessible.

2. The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT statement, to be made available for processing in the records area associated with the sort-file.

3. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rule for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

   When the INTO phrase is used, the data is available in both the input record area and the data area associated with identifier.

4. After execution of the imperative statement in the AT END phrase, no RETURN statement may be executed within the current output procedure.

7.33 THE SEARCH STATEMENT

7.33.1 FUNCTION

The SEARCH statement is used to search a table for a table-element that satisfies the specified condition and to adjust the associated index-name to indicate that table-element.

7.33.2 GENERAL FORMAT

Format 1

$$\underline{\text{SEARCH}}\ \text{identifier-1}\ \left[\underline{\text{VARYING}}\ \left\{\begin{array}{l}\text{index-name-1}\\ \text{identifier-2}\end{array}\right\}\right]$$

$$[\text{; AT}\ \underline{\text{END}}\ \text{imperative-statement-1}]$$

$$\text{; }\underline{\text{WHEN}}\ \text{condition-1}\ \left\{\begin{array}{l}\text{imperative-statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}$$

$$\left[\text{; }\underline{\text{WHEN}}\ \text{condition-2}\ \left\{\begin{array}{l}\text{imperative-statement-3}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}\right]\ldots$$

Format 2

$$\underline{\text{SEARCH}}\ \underline{\text{ALL}}\ \text{identifier-1}\ \ [\text{; AT}\ \underline{\text{END}}\ \text{imperative-statement-1}]$$

$$\text{; }\underline{\text{WHEN}}\ \text{condition-1}\ \left\{\begin{array}{l}\text{imperative-statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}$$

7.33.3 SYNTAX RULES

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in Format 2 must also contain the KEY IS option in its OCCURS clause.

2. Identifier-2, when specified, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point. Identifier-2 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

3. In Format 1, condition-1, condition-2, etc., may be any condition as described in 7.4, Conditions.

4. In Format 2, condition-1 may consist of a relation condition incorporating the relation EQUALS or EQUAL TO or equal sign, or a condition-name condition, where the VALUE clause that describes the condition-name contains only a single literal. Alternatively, condition-1 may be a compound condition formed from simple conditions of the type just mentioned, with AND as the only connective. Any data-name that appears in the KEY clause of identifier-1 may appear as the subject or object of a test or be the name of the conditional variable with which the tested condition-name is associated; however, all preceding data-names in the KEY clause must also be included within condition-1. No other tests may appear within condition-1.

7.33.4 GENERAL RULES

1. If Format 1 of the SEARCH is used, a serial type of search operation takes place, starting with the current index setting.

   a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next sentence.

   b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1 the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element which exceeds the last element of the table by one or more occurrences, in which case the search terminates as indicated in 1a above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

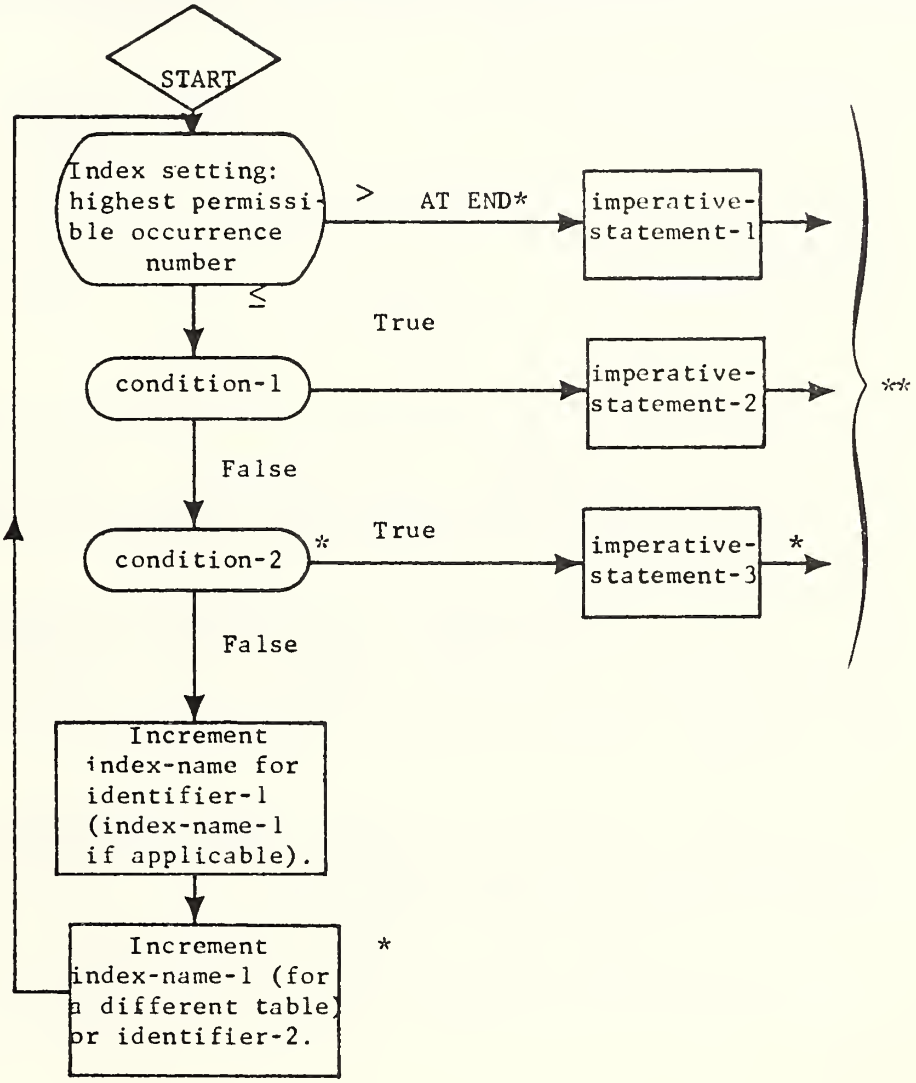
2. If Format 2 of the SEARCH is used, a nonserial type of search operation may take place in which case, the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation in a manner specified by the implementor, with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the table, or that is less than the value that corresponds to the first element of the table. If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END phrase appears, or to the next sentence when this phrase does not appear; in either case the final setting of the index is not predictable. If condition-1 can be satisfied, index indicates an occurrence that allows condition-1 to be satisfied, and control passes to imperative-statement-2.

3. After execution of imperative-statement-1, imperative-statement-2, or imperative-statement-3, that does not terminate with a GO TO statement, control passes to the next sentence.

4. In the VARYING index-name-1 phrase, if index-name-1 appears in the INDEXED BY clause of identifier-1, that index-name is used for this search, otherwise the first (or only) index-name given in the INDEXED BY clause of identifier-1 is used. If index-name-1 appears in the INDEXED BY clause of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

5. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two or three dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire two or three dimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

A diagram of the Format 1 SEARCH operation containing two WHEN phrases follows.


START
Index setting: highest permissible occurrence number
>
AT END*
≤
imperative-statement-1
True
condition-1
imperative-statement-2
False
condition-2
*
True
imperative-statement-3
*
False
Increment index-name for identifier-1 (index-name-1 if applicable).
Increment index-name-1 (for a different table) or identifier-2.
*
**

* These operations are options included only when specified in the SEARCH statement.

** Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

7.34 THE SEEK STATEMENT

7.34.1 FUNCTION

The SEEK statement initiates the accessing of a mass storage data record for subsequent reading or writing.

7.34.2 GENERAL FORMAT

SEEK file-name RECORD

7.34.3 GENERAL RULES

1. Any restrictions in the use or implementation of the SEEK statement must be specified by the implementor.

2. A SEEK statement pertains only to mass storage files in the random access mode and may be executed prior to the execution of each READ and WRITE statement.

3. The SEEK statement uses the contents of the data-name in the ACTUAL KEY clause (see 5.4.1, The FILE-CONTROL Paragraph) for the location of the record to be accessed.

4. Two SEEK statements for the same mass storage file may logically follow each other.

5. A SEEK statement is related to a subsequent READ or WRITE statement only if they are all in the in-line procedure or are all in the same processing cycle of an out-of-line procedure.

7.35 THE SET STATEMENT

7.35.1 FUNCTION

The SET statement establishes reference points for table-handling operations by setting index-names associated with table elements.

7.35.2 GENERAL FORMAT

Format 1

$$\underline{\text{SET}} \left\{ \begin{array}{l} \text{index-name-1 [, index-name-2]...} \\ \\ \text{identifier-1 [, identifier-2]...} \end{array} \right\} \underline{\text{TO}} \left\{ \begin{array}{l} \text{index-name-3} \\ \text{identifier-3} \\ \text{literal-1} \end{array} \right\}$$

Format 2

$$\underline{\text{SET}} \text{ index-name-4 [, index-name-5]...} \left\{ \begin{array}{l} \underline{\text{UP}}\ \underline{\text{BY}} \\ \underline{\text{DOWN}}\ \underline{\text{BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{literal-2} \end{array} \right\}$$

7.35.3 SYNTAX RULES

1. All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2 and index-name-5, respectively.

7.35.4 GENERAL RULES

1. All identifiers must name either index data items, or elementary items described as an integer, except that identifier-4 in Format 2 must not name an index data item. When a literal is used, it must be a positive integer. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

2. In Format 1, the following action occurs:

    a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element refered to by index-name-3, identifier-3, or literal-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place. If the value contained in an index data item does not correspond to an occurrence number of an element in the table indexed by index-name-1, the result is undefined.

    b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index data item. Literal-1 cannot be used in this case.

    c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor literal-1 can be used in this case.

    d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time, the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

3. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of literal-2 or identifier-4; thereafter, the process is repeated for index-name-5, etc. Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.

7.36 THE SORT STATEMENT

7.36.1 FUNCTION

The SORT statement creates a sort-file by executing input procedures or by transferring records from another file, sorts the records in the sort-file on a set of specified keys, and in the final phase of the sort operation, makes available each record from the sort-file, in sorted order, to some output procedures or to an output file.

7.36.2 GENERAL FORMAT

```
SORT file-name-1 ON { DESCENDING } KEY data-name-1  [, data-name-2 ]...
                    { ASCENDING  }

           [ ; ON { DESCENDING } KEY data-name-3  [, data-name-4 ]... ]...
                  { ASCENDING  }

{ INPUT PROCEDURE IS section-name-1 [{ THRU    } section-name-2 ] }
{                                   [{ THROUGH }                ] }
{ USING file-name-2                                               }

{ OUTPUT PROCEDURE IS section-name-3 [ { THRU    }  section-name-4 ] }
{                                    [ { THROUGH }                 ] }
{ GIVING file-name-3                                                 }
```

7.36.3 SYNTAX RULES

1. File-name-1 must be described in a Sort File Description entry in the Data Division. Each data-name must represent data items described in records associated with file-name-1.

2. Section-name-1 represents the name of an input procedure. Section-name-3 represents the name of an output procedure.

3. File-name-2 and file-name-3 must be described in a File Description entry, not in a Sort File Description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2 and file-name-3 must be equal to the actual size of the logical record(s) described for file-name-1. If the data description of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause equal amounts of computer storage to be allocated for the corresponding records.

4. The data-names may be qualified. However, if the same data-name is used in describing the same key in more than one record description it need not be qualified when used in the SORT statement.

5. The words THRU and THROUGH are equivalent.

7.36.4 GENERAL RULES

1. Programs that contain SORT statements are divided into two classes: basic sorting and extended sorting.

   a. The Procedure Division of a basic sort program contains one SORT statement and a STOP RUN statement in the first nondeclarative portion. Other sections consist only of input and output procedures associated with the SORT statement.

   b. The Procedure Division of an extended sort program may contain more than one SORT statement appearing anywhere except:

      (1) in the Declaratives portion, or

      (2) in the input and output procedures associated with a SORT statement.

2. The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY clauses. In the format data-name-1 is the major key, data-name-2 is the next most significant key, etc. The direction of the sort depends on the use of the ASCENDING or DESCENDING clauses as follows:

   a. When an ASCENDING clause is used, the sorted sequence will be from the lowest value of key to the highest value according to the rules for comparison of operands in a relation condition.

   b. When a DESCENDING clause is used, the sorted sequence will be from the highest value of key to the lowest value according to the rules for comparison of operands in a relation condition.

3. The record description for every record that is a logical record associated with the sort-file description must contain the KEY items data-name-1, data-name-2, etc. These KEY items are subject to the following rules:

   a. They may not be variable length items.

   b. Where more than one record description appears, the key items need only be described in one of the record descriptions. When the key items are described in more than one record the data descriptions must be equivalent and their starting position must always be the same number of character positions from the beginning of each record.

   c. They may not contain an OCCURS clause, nor be subordinate to entries that contain an OCCURS clause.

4. The input procedure, if present, must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file. Control must not be passed to the input procedure except when a related SORT statement is being executed, because the RELEASE statements in the input procedure have no meaning unless they are controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

   a. The input procedure must not contain any SORT statements.

   b. The input procedure must not contain any explicit transfers of control to points outside the input procedure; ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. PROCESS and CALL statements are not permitted in an input procedure. COBOL statements are allowed that will cause an implied transfer of control by the compiler to Declaratives for label procedures, error procedures and reporting procedures.

   c. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedures; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

5. If an input procedure is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

6. The output procedure, if present, must consist of one or more sections that appear contiguously in a source program and do not form part of any input procedure. The output procedure must include as least one RETURN statement in order to make sorted records available for processing. Control must not be passed to the output procedure except when a related SORT statement is being executed, because the RETURN statements in the output procedure have no meaning unless they are controlled by a SORT statement. The output procedure may consist of any procedures needed to select, modify or copy the records that are being returned one at a time in sorted order, from the sort-file. The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must not contain any SORT statements.

   b. The output procedure must not contain any transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. PROCESS and CALL statements are not permitted in an output procedure. COBOL statements are allowed that will cause an implied transfer of control by the compiler to Declaratives for label procedures and reporting procedures.

   c. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

7. If an output procedure is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

8. Segmentation as defined in Chapter 8 can be applied to the sections within the input or output procedures.

9. If the USING option is specified, all the records in file-name-2 are transferred automatically to the file-name-1. At the time of execution of the SORT statement, file-name-2 must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2. These implicit functions are performed such that any associated USE procedures are executed. The terminating function is performed as if the CLOSE statement had been explicitly written without optional phrases. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 to the file area for file-name-1 and the release of records to the initial phase of the sort operation.

10. If the GIVING option is used, all the sorted records in file-name-1 are automatically transferred to file-name-3 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-3 must not be open. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of file name-3. These implicit functions are performed such that any associated USE procedures are executed. The terminating function is performed as if the CLOSE statement had been written without optional phrases. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and the moving of the records from the file area for file-name-1 to the file area for file-name-3.

7.37 THE STOP STATEMENT

7.37.1 FUNCTION

The STOP statement causes a permanent or temporary suspension of the execution of the run unit.

7.37.2 GENERAL FORMAT

$$\underline{\text{STOP}} \left\{ \begin{array}{l} \underline{\text{RUN}} \\ \text{literal} \end{array} \right\}$$

7.37.3 SYNTAX RULES

1. The literal may be numeric or nonnumeric or may be any figurative constant, except ALL.

7.37.4 GENERAL RULES

1. If the RUN phrase is used, then the ending procedure established by the installation and/or the compiler is instituted.
2. If the literal option is used, the literal is communicated to the operator. Continuation of the object program begins with the execution of the next statement in sequence.
3. If a STOP RUN statement appears in an imperative sentence, then it must appear as the only or last statement in a sequence of imperative statements.

7.38 THE SUBTRACT STATEMENT

7.38.1 FUNCTION

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

7.38.2 GENERAL FORMAT

Format 1

$$\underline{\texttt{SUBTRACT}}\ \left\{\begin{array}{l}\texttt{literal-1}\\ \texttt{identifier-1}\end{array}\right\}\ \left[\begin{array}{l}\texttt{, literal-2}\\ \texttt{, identifier-2}\end{array}\right]\ \ldots\ \underline{\texttt{FROM}}\ \texttt{identifier-m}\ [\underline{\texttt{ROUNDED}}]$$

$$[\texttt{, identifier-n}\ [\underline{\texttt{ROUNDED}}]\ ]\ldots\ [\texttt{; ON}\ \underline{\texttt{SIZE}}\ \underline{\texttt{ERROR}}\ \texttt{imperative-statement}]$$

Format 2

$$\underline{\texttt{SUBTRACT}}\ \left\{\begin{array}{l}\texttt{literal-1}\\ \texttt{identifier-1}\end{array}\right\}\ \left[\begin{array}{l}\texttt{, literal-2}\\ \texttt{, identifier-2}\end{array}\right]\ \ldots\ \underline{\texttt{FROM}}\ \left\{\begin{array}{l}\texttt{literal-m}\\ \texttt{identifier-m}\end{array}\right\}$$

$$\underline{\texttt{GIVING}}\ \texttt{identifier-n}\ [\underline{\texttt{ROUNDED}}]\ [\texttt{, identifier-o}\ [\underline{\texttt{ROUNDED}}]\ ]\ldots$$

$$[\texttt{; ON}\ \underline{\texttt{SIZE}}\ \underline{\texttt{ERROR}}\ \texttt{imperative-statement}]$$

Format 3

$$\underline{\texttt{SUBTRACT}}\ \left\{\begin{array}{l}\underline{\texttt{CORR}}\\ \underline{\texttt{CORRESPONDING}}\end{array}\right\}\ \texttt{identifier-1}\ \ \underline{\texttt{FROM}}\ \ \texttt{identifier-2}\ [\underline{\texttt{ROUNDED}}]$$

$$[\texttt{; ON}\ \underline{\texttt{SIZE}}\ \underline{\texttt{ERROR}}\ \texttt{imperative-statement}]$$

7.38.3 SYNTAX RULES

1. Each identifier must refer to a numeric elementary item except in Format 2, where any identifiers that appear only to the right of the word GIVING may refer to data items that contain editing symbols.

2. The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data items that follow the word GIVING, aligned on their decimal points, must not contain more than eighteen digits.

3. CORR is an abbreviation for CORRESPONDING.

7.38.4 GENERAL RULES

1. See 7.6.1, The ROUNDED Option; 7.6.2, The SIZE ERROR Option; 7.6.3, The CORRESPONDING Option; 7.6.6, Multiple Results in Arithmetic Statements; and 3.2.1.2.5, Special Registers.

2. In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m, identifier-n, etc., and the differences are stored as the new value of identifier-m, identifier-n, etc.

3. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m and the result of the subtraction is stored as the new value of identifier-n, identifier-o, etc.

4. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

5. The compiler insures enough places are carried so as not to lose significant digits during execution.

7.39 THE SUSPEND STATEMENT

7.39.1 FUNCTION

The SUSPEND statement stops the release of subsequent logical records to a graphic display device until re-initiated by the operator.

7.39.2 GENERAL FORMAT

$$\underline{\text{SUSPEND}} \quad \left\{ \begin{array}{ll} \text{file-name-1} & \begin{bmatrix}\text{literal-1}\\ \text{identifier-1}\end{bmatrix} \\ \text{report-name-1} & \begin{bmatrix}\text{literal-2}\\ \text{identifier-2}\end{bmatrix} \end{array} \right\} \left[, \left\{ \begin{array}{ll} \text{file-name-2} & \begin{bmatrix}\text{literal-3}\\ \text{identifier-3}\end{bmatrix} \\ \text{report-name-2} & \begin{bmatrix}\text{literal-4}\\ \text{identifier-4}\end{bmatrix} \end{array} \right\} \ldots \right]$$

7.39.3 SYNTAX RULES

1. The figurative constant ALL must not be used as literal-1, literal-2, etc.

7.39.4 GENERAL RULES

1. When file-name-1, or the file-name associated with a report-name, is assigned to a graphic display device, the SUSPEND statement releases a logical record to the file named. When this logical record is transmitted to the graphic display device, the further transmission of logical records to the graphic display device is suspended until re-initiated by the graphic display device operator. When the file-name or the file-name associated with a report-name, is not assigned to a graphic display device, the effect of the SUSPEND statement is defined by the implementor.

2. An OPEN statement (with no intervening CLOSE statement for the file) must be executed for a file prior to executing the SUSPEND statement for that file.

3. The object program will stop execution only when the release of logical records to the output device is not possible until re-initiated by the operator.

4. The implementor will specify the means of re-initiating the release of logical records to the graphic display device.

5. The value of literal-1, literal-2, identifier-1, identifier-2, etc., is written on the implementor's standard display device. When more than one such device is present, the one most logically associated with the graphic display device will be used.

6. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

7. The implementor will define, for each hardware device, the size of a data transfer.

8. If the hardware device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred.

9. If a hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:
   a. If the size of the data item being transferred exceeds the size of the data that the hardware device is capable of receiving in a single transfer, the data beginning with the left most character is stored aligned to the left in the receiving hardware device, and additional data is requested.
   b. If the size of the data item that the hardware device is capable of receiving exceeds the size of the data item being transferred, the transferred data is stored aligned to the left in the receiving hardware device.

7.40 THE TERMINATE STATEMENT

7.40.1 FUNCTION

The TERMINATE statement terminates the processing of a report.

7.40.2 GENERAL FORMAT

```
TERMINATE   { report-name-1  [ , report-name-2 ] ...}
            { ALL                                   }
```

7.40.3 SYNTAX RULES

1. Each report-name given in a TERMINATE statement must be defined by a RD entry in the Data Division.

7.40.4 GENERAL RULES

1. The TERMINATE statement produces all the control footings associated with this report as if a control break had just occurred at the highest level, and completes the Report Writer functions for the named reports. The TERMINATE statement also produces the last PAGE FOOTING and the REPORT FOOTING report groups associated with this report.

2. Appropriate PAGE and OVERFLOW HEADING and/or FOOTING report groups are prepared in their respective order for the report description

3. If ALL is specified, all report-names defined in the Report Section of the Data Division which were initiated are terminated.

4. A second TERMINATE for a particular report may not be executed unless a second INITIATE statement has been executed for the report-name. If a TERMINATE statement has been executed for a report, a GENERATE statement for that report must not be executed unless an intervening INITIATE statement for that report is executed.

5. The TERMINATE statement does not close the file with which the report is associated; a CLOSE statement for the file must be given by the user. However, the associated file must be open at the time the TERMINATE statement is executed. The TERMINATE statement performs Report Writer functions for individually described report programs analogous to the input/output functions that the CLOSE statement performs for individually described files.

6. SOURCE clauses used in TYPE CONTROL FOOTING FINAL or TYPE REPORT FOOTING report groups refer to the values of the items at the execution time of the TERMINATE statement.

7.41 THE USE STATEMENT

7.41.1 FUNCTION

1. The USE statement specifies procedures for input-output label and error handling that are in addition to the standard procedures provided by the input output system. It is also used to specify Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

2. The USE statement is also necessary in specifying out-of-line procedural statements for processing mass storage files.

7.41.2 GENERAL FORMAT

Format 1

$$\underline{\text{USE}}\ \underline{\text{AFTER}}\ \text{STANDARD}\ \underline{\text{ERROR}}\ \underline{\text{PROCEDURE}}\ \text{ON} \left\{\begin{array}{l} \text{file-name-1}\ [\ ,\ \text{file-name 2}\ ]\ \ldots \\ \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \\ \underline{\text{I-O}} \end{array}\right\} .$$

Format 2

$$\underline{\text{USE}} \left\{\begin{array}{l} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{array}\right\} \text{STANDARD} \left[\begin{array}{l} \underline{\text{BEGINNING}} \\ \underline{\text{ENDING}} \end{array}\right] \left[\begin{array}{l} \underline{\text{REEL}} \\ \underline{\text{FILE}} \\ \underline{\text{UNIT}} \end{array}\right]$$

$$\underline{\text{LABEL}}\ \underline{\text{PROCEDURE}}\ \text{ON} \left\{\begin{array}{l} \text{file-name-1}\ [\ ,\ \text{file-name-2}\ ]\ \ldots \\ \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \\ \underline{\text{I-O}} \end{array}\right\} .$$

Format 3

$$\underline{\text{USE}}\ \underline{\text{BEFORE}}\ \underline{\text{REPORTING}}\ \text{identifier-1}\ [\ ,\ \text{identifier-2}\ ]\ \ \ldots\ \ .$$

Format 4

$$\underline{\text{USE}}\ \text{FOR}\ \underline{\text{RANDOM}}\ \text{PROCESSING.}$$

7.41.3 SYNTAX RULES

1. A USE statement, when present, must immediately follow a section header in the Declarative portion of the Procedure Division and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

2. If the file-name phrase is used as a part of Format 2, the File Description entry for file-name-1 must not specify a LABEL RECORDS ARE OMITTED clause.

3. In Format 3, identifier-1 represents a report group named in the Report Section of the Data Division. An identifier must not appear in more than one USE statement.

   No Report Writer statement (GENERATE, INITIATE, or TERMINATE) may be written in a procedural paragraph or paragraphs following the USE sentence in the Declarative portion.

4. When using Format 4 (for mass storage only), a PROCESS statement cannot appear in a USE FOR RANDOM PROCESSING section.

5. The USE statement itself is never executed; rather, it defines the conditions calling for the execution of the USE procedures.

6. If the words BEGINNING or ENDING are not included in Format 2, the designated procedures are executed for both beginning and ending labels.

   If neither UNIT, REEL, nor FILE is included, the designated procedures are executed for both REEL or UNIT, whichever is appropriate, and FILE labels. The REEL phrase is not applicable to mass storage files. The UNIT phrase is not applicable to files in the random access mode.

7. The same file-name can appear in a different specific arrangement of a Format. However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

8. No file-name may represent a sort-file.

7.41.4 GENERAL RULES

1. The designated procedures are executed by the input-output system at the appropriate time as follows:

    a. In Format 1, after completing the standard input-output error routine.

    b. In Format 2, before or after a beginning or ending input label check procedure is executed.

    Before a beginning or ending output label is created.

    After a beginning or ending output label is created, but before it is written.

    Before or after a beginning or ending input-output label check procedure is executed.

2. In Format 2, within the procedures of a USE declarative in which the USE statement specifies a phrase other than the file-name-1 phrase references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but at the same time, does not appear in any data record of this program. Furthermore, a common label item must have the same name, description, and relative position in every label record.

    If the INPUT, OUTPUT, or I-O option is specified, the USE procedures do not apply respectively to input, output, or input-output files that are described with the LABEL RECORDS ARE OMITTED clause.

3. In Format 3, the designated procedures are executed by the Report Writer just before the named report is produced, regardless of page, overflow, or control break associations with report groups. The report group may be any type except DETAIL.

4. Format 4 is used for an out-of-line procedure that can be executed asynchronously. The execution of a PROCESS statement causes the processing of a specified out-of-line procedure to be initiated.

5. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE declarative having Formats 1, 2, 3 or to the procedures associated with such a USE declarative.

6. In the statements of the out-of-line procedures, the execution of an ALTER or PERFORM statement affects that particular processing cycle only; that is, any other asynchronous processing cycle in the out-of-line set of procedural statements is not affected by another concurrent processing cycle in which an ALTER or PERFORM statement is executed.

7.42 THE WRITE STATEMENT

7.42.1 FUNCTION

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of a printer. For mass storage files, the WRITE statement also allows the performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item are found to be invalid.

7.42.2 GENERAL FORMAT

Format 1

$$\texttt{\underline{WRITE} record-name [\underline{FROM} identifier-1]}$$

$$\left[\left\{\begin{array}{l}\texttt{\underline{BEFORE}}\\ \\ \texttt{\underline{AFTER}}\end{array}\right\}\texttt{ ADVANCING }\left\{\begin{array}{l}\texttt{identifier-2}\\ \texttt{integer}\\ \texttt{mnemonic-name}\end{array}\right.\quad\left.\begin{array}{l}\texttt{LINES}\\ \texttt{LINES}\\ \\ \end{array}\right\}\right]$$

$$\left[\texttt{; AT }\left\{\begin{array}{l}\texttt{\underline{END-OF-PAGE}}\\ \\ \texttt{\underline{EOP}}\end{array}\right\}\texttt{ imperative-statement}\right]$$

Format 2

$$\texttt{\underline{WRITE} record-name [\underline{FROM} identifier-1]}$$

$$\texttt{; \underline{INVALID} KEY imperative-statement}$$

7.42.3 SYNTAX RULES

1. An OPEN statement for a file must be executed prior to executing the first WRITE statement for that file.

2. Record-name and identifier-1 must not refer to the same storage area.

3. When the mnemonic-name option is used, the name is identified with a particular feature specified by the implementor. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the Environment Division.

4. The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

5. When identifier-2 or integer is used in the ADVANCING option, it must be the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

6. Record-names must not be part of a sort-file.

7. If the END-OF-PAGE option is used, the LINAGE clause must be present in the FD entry for the associated file.

7.42.4 GENERAL RULES

1. Format 2 is used for processing mass storage files.

2. The logical record released by the execution of the WRITE statement is no longer available unless the associated file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated output file.

3. If the FROM phrase is specified the data is moved from the area specified by identifier to the output area, according to the rules specified for the MOVE statement without the CORRESPONDING phrase. After execution of the WRITE statement is completed, the information in identifier-1 is available, even though that in record-name is not.

4. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each record on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided by the implementor so as to cause single spacing. If the ADVANCING phrase is used, automatic advancing is overridden:

   a. If identifier-2 is specified, the printer page is advanced the number of lines equal to the current value associated with identifier-2.

   b. If integer is specified, the printer page is advanced the number of lines equal to the value of integer.

   c. If mnemonic-name is specified, the printer page is advanced according to the rules specified by the implementor for that hardware device. If mnemonic-name is specified with the identifier or integer phrase of the LINAGE clause in the File Description entry the results may be unpredictable.

   d. If the BEFORE phrase is used, the record is printed before the printer page is advanced according to rules a, b, and c above.

   e. If the AFTER phrase is used, the record is printed after the printer page is advanced according to rules a, b, and c above.

5. If the logical end of the printer page is reached during the execution of a Format 1 WRITE statement with the END-OF-PAGE phrase the imperative-statement specified in the END-OF-PAGE clause is executed. The END-OF-PAGE limit is specified in the LINAGE clause in the File Section of the Data Division.

6. If an END-OF-PAGE is reached during the execution of a WRITE statement with both the ADVANCING and END-OF-PAGE phrase the WRITE and ADVANCING

operation is executed prior to executing the imperative-statement in the END-OF-PAGE clause.

7. For mass storage files in the sequential access mode, the imperative-statement in the INVALID KEY clause is executed when the end of the last segment of the file is reached and an attempt is made to execute a WRITE statement for that file. The last segment of a file is specified in the FILE LIMITS clause or in the ASSIGN clause of the Environment Division.

8. For files in the random access mode, the WRITE statement implicitly performs the function of the SEEK statement for a specific mass storage record, unless a SEEK statement is executed for this record prior to the execution of the WRITE statement. A SEEK statement is related to a subsequent WRITE statement only if both are in the in-line procedure or both are in the same processing cycle of an out-of-line procedure. The imperative-statement in the INVALID KEY phrase is executed when the contents of the ACTUAL KEY being used to obtain the mass storage record is found to be invalid. When an INVALID KEY condition exists, no writing takes place and the information in the record area is available.

9. After the recognition of an end-of-reel or an end-of-unit of an OUTPUT or I-O mass storage file in the sequential access mode that is contained on more than one physical mass storage unit, the WRITE statement performs the following operations:

   a. The standard ending reel/unit label procedure and the user's ending reel/unit label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

   b. A reel/unit swap.

   c. The standard beginning reel/unit label procedures and the user's beginning reel/unit label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

10. If a mass storage file (either OUTPUT or I-O), in the random access mode is contained on more than one physical mass storage unit, and not all of the physical units are simultaneously available, the procedures for making the physical units available are specified by the implementor See 5.4.1, The FILE-CONTROL Paragraph.

CHAPTER 8

SEGMENTATION

8.1 GENERAL DESCRIPTION

COBOL segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements.

8.1.1 SCOPE

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division and the Environment Division are considered in determining segmentation requirements for an object program.

8.1.2 ORGANIZATION

8.1.2.1 Program Segments

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to insure uniqueness.

8.1.2.2 Fixed Portion

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of two types of segments: permanent segments and overlayable fixed segments.

A permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program. An overlayable fixed segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlaid by another segment to optimize memory utilization. Variation of the number of permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT clause (see 8.2.2, SEGMENT-LIMIT). Such a segment, if called for by the program, is always made available in its last used state.

8.1.2.3 Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, either an overlayable fixed segment or another independent segment. An independent segment is effectively in its initial state each time the segment is made available to the program.

8.1.3 SEGMENT CLASSIFICATION

Sections which are to be segmented are classified, using a system of priority-numbers (see 8.2.1 below) and the following criteria:

1. Logic Requirements - Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections which are used less frequently are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.

2. Frequency of Use - Generally, the more frequently a section is referred to, the lower its priority-number; the less frequently it is referred to, the higher its priority-number.

3. Relationship to Other Sections - Sections which frequently communicate with one another should be given the same priority-numbers.

8.1.4 SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. If any reordering of the object program is required to handle the flow from segment to segment, according to the rules in 8.2.1, the implementor must provide control transfers to maintain the logical flow specified in the source program. The implementor must also provide all controls necessary for a segment to operate whenever the segment is used. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

8.2 STRUCTURE OF PROGRAM SEGMENTS

8.2.1 PRIORITY-NUMBERS

Section classification is accomplished by means of a system of priority-numbers. The priority-number is included in the section header.

8.2.1.1 General Format

section-name SECTION [priority-number].

8.2.1.2 Syntax Rules

1. The priority-number must be an integer ranging in value from 0 through 99.

2. If the priority-number is omitted from the section header, the priority is assumed to be 0.

8.2.1.3 General Rules

1. All sections which have the same priority-number constitute a program segment with that priority.

2. Segments with priority-number 0 through 49 belong to the fixed portion of the object program.

3. Segments with priority-number 50 through 99 are independent segments.

4. Sections in the Declaratives must not contain priority-numbers in their section headings. These sections are defined to have a priority of 0.

8.2.2 SEGMENT-LIMIT

Ideally, all program segments having priority-numbers ranging from 0 through 49 should be specified as permanent segments. However, when insufficient memory is available to contain all permanent segments plus the largest overlayable segment, it becomes necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means by which he can reduce the number of permanent segments in his program, while still retaining the logical properties of fixed portion segments (priority-numbers 0 through 49).

8.2.2.1 General Format

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph and has the following format:

[, SEGMENT-LIMIT IS priority-number]

8.2.2.2 Syntax Rules

1. Priority-number must be an integer ranging in value from 1 through 49.

2. When the SEGMENT-LIMIT clause is specified, only those segments having priority-numbers from 0 up to, but not including, the priority-number designated as the segment limit, are considered as permanent segments of the object program.

3. Those segments having priority-numbers from the segment limit through 49 are considered as overlayable fixed segments.

4. When the SEGMENT-LIMIT clause is omitted, all segments having priority-numbers from 0 through 49 are considered as permanent segments of the object program.

CHAPTER 9

THE COBOL LIBRARY

9.1 INTRODUCTION

The COBOL library contains text that is available to a source program at compile time. The effect of the compilation of library text is the same as if the text were actually written as part of the source program.

The COBOL library may contain text for the Environment Division, the Data Division and the Procedure Division available through the use of the COPY statement.

9.2 THE COPY STATEMENT

9.2.1 GENERAL FORMAT

$$\underline{\text{COPY}}\ \text{library-name}$$

$$\left[\ \underline{\text{REPLACING}}\ \left\{\begin{array}{l}\text{word-1}\\ \text{identifier-1}\end{array}\right\}\ \underline{\text{BY}}\ \left\{\begin{array}{l}\text{word-2}\\ \text{identifier-2}\end{array}\right\}\right.$$

$$\left.\left[\ ,\ \left\{\begin{array}{l}\text{word-3}\\ \text{identifier-3}\end{array}\right\}\ \underline{\text{BY}}\ \left\{\begin{array}{l}\text{word-4}\\ \text{identifier-4}\end{array}\right\}\right]\ldots\ \right].$$

9.2.2 SYNTAX RULES

1. A word is any COBOL word in a library routine that is not a reserved word.

2. Replacement of one identifier by another includes all associated qualifiers, subscripts, and indexes.

9.2.3 GENERAL RULES

1. The COPY statement may appear as follows:
   a. in any of the paragraphs in the Environment Division.

   b. in any level indicator entries or an 01 level-number entry in the Data Division.
   c. in a section or a paragraph in the Procedure Division.

2. No other statement or clause may appear in the same entry as the COPY statement.

3. The library text is copied from the library and the result of the compilation is the same as if the text were actually a part of the source program.

4. The copying process is terminated by the end of the library text.

5. If the REPLACING phrase is used, each occurrence of word-1, word-3, identifier-1, identifier-3, etc., in the text being copied from the library is replaced by the word or identifier associated with it in the REPLACING phrase.

6. Use of the REPLACING option does not alter the material as it appears on the library.

7. The implementor determines whether the COPY statement itself or the statements of the library text to which it refers, or both, is to appear on the output listing. If both, the relationship between the two must be clearly indicated.

8. The text contained on the library must not contain any COPY statements.

CHAPTER 10

REFERENCE FORMAT

10.1 GENERAL DESCRIPTION

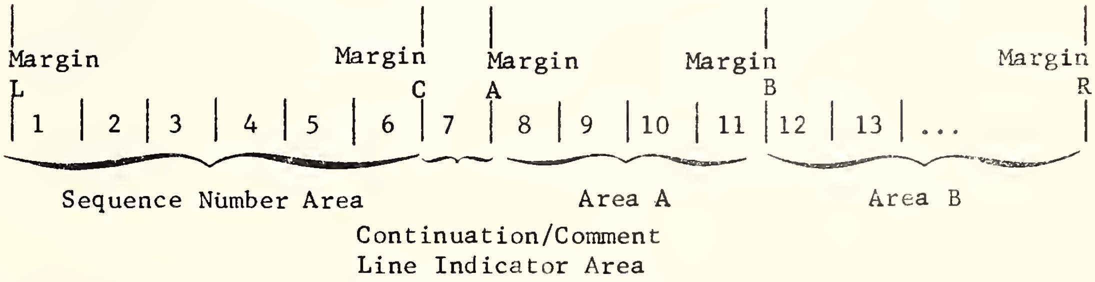
The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions in a line on an input-output medium. Each implementor must define what is meant by lines and character positions for each input-output medium used with his compiler. Within these definitions, each COBOL compiler accepts source programs written in reference format and produces an output listing in reference format.

The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The division of a source program must be ordered as follows: the Identification Division, then the Environment Division, then the Data Division, then the Procedure Division. Each division must be written according to the rules for the reference format.

10.2 REFERENCE FORMAT REPRESENTATION

The reference format for a line is represented as follows:



Margin L designates the left-most character position of a line.

Margin C designates the seventh character position relative to L.

Margin A designates the eighth character position relative to L.

Margin B designates the twelfth character position relative to Margin L.

Margin R designates the right-most character position of a line.

The sequence number area occupies the six character positions beginning at Margin L.

The continuation/comment line indicator area occupies one character position beginning at Margin C.

Area A occupies four character positions beginning at Margin A.

Area B occupies a finite number of character positions specified by the implementor beginning at Margin B.

10.2.1 SEQUENCE NUMBERS

A sequence number, consisting of six digits in the sequence number area, may be used to label a source program line.

10.2.2 CONTINUATION OF LINES

Any sentence or entry that requires more than one line is continued by starting subsequent line(s) in Area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line.

A hyphen in the continuation area of a line indicates that the first nonblank character in Area B of the current line is the successor of the last nonblank character of the preceding line without any intervening space. However, if the continued line contains a nonnumeric literal without closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continued line must be blank.

If there is no hyphen in the continuation area of a line, it is assumed that the last character in the preceding line is followed by a space.

10.2.3 BLANK LINES

A blank line is one that is blank from Margin C to Margin R, inclusive. A blank line can appear anywhere in the source program, except immediately preceding a continuation line. (See 10.2.2, Continuation of Lines.)

10.3 DIVISION, SECTION, PARAGRAPH FORMATS

10.3.1 DIVISION HEADER

The division header starts in Area A with the division-name, is followed by a space, then the word DIVISION, then optionally the PREPARED FOR clause in the case of the Data Division only (see 6.1.2.2, Data Division Structure), then a period. After the division header, no text may appear before the following section header or paragraph header or paragraph-name (that is before the next line without a hyphen or asterisk in the continuation area and with a nonblank character in Area A), except that the PREPARED FOR clause may be present after the Data Division header.

10.3.2 SECTION HEADER

The name of a section starts in Area A of any line except the first line of a division reference format, is followed by a space, then the word SECTION, then optionally a space followed by a priority-number, then a period followed by a space. After the section header, no text may appear before the following paragraph header or paragraph-name (that is before the next line without a hyphen or asterisk in the continuation area and with a nonblank character in Area A), with the exception of the COPY and USE sentences.

A section consists of paragraphs in the Environment and Procedure Divisions and Data Division entries in the Data Division.

10.3.3 PARAGRAPH HEADER, PARAGRAPH-NAME AND PARAGRAPH

A paragraph consists of a paragraph-name followed by one or more sentences, or a paragraph header followed by one or more entries. A paragraph header starts in Area A of any line following the first line of a division or a section.

The name of a paragraph starts in Area A of any line following the first line of a division or a section and ends with a period followed by a space.

The first sentence or entry in a paragraph begins in Area B of either the same line as the paragraph-name or paragraph header or the next nonblank line that is not a comment line. Successive sentences or entries either begin in Area B of the same line as the preceding sentence or entry or in Area B of the next nonblank line that is not a comment line.

A sentence consists of one or more statements, an entry consists of one or more clauses; all sentences and entries must be followed by a period followed by a space.

When the sentences or entries of a paragraph require more than one line they may be continued as described in 10.2.2, Continuation of Lines.

10.4 DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or a level-number, followed by a space, followed by the name of a data item (except in the Report Section), followed by a sequence of independent clauses describing the data item. Each clause, except the last clause of an entry, may be terminated by a semicolon followed by a space. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level-number.

A level indicator is any of the following: FD, SD, RD, SA.

In those Data Division entries that begin with a level indicator, the level indicator begins in Area A followed in Area B by its associated data-name and appropriate descriptive information.

Those Data Division entries that begin with level-numbers are called data description entries.

In those data description entries that begin with a level-number 01 or 77, the level-number begins in Area A followed in Area B by its associated record-name or item-name and appropriate descriptive information.

A level-number may be one of the following set: 01 through 49, 66, 77, 88. Single digit level-numbers are written either as a single space followed by a digit or as a zero followed by a digit. At least one space must separate a level-number from the word following the level-number.

Successive data description entries may have the same format as the first or may be indented according to level-number. The entries in the output listing need be indented only if the input is indented. Indentation does not affect the magnitude of a level-number.

When level-numbers are to be indented, each new level-number may begin any number of spaces to the right of Margin A. The extent of indentation to the right is determined only by the width of the physical medium.

10.5 DECLARATIVES

The key word DECLARATIVES and the key words END DECLARATIVES that precede and follow, respectively, the Declaratives portion of the Procedure Division must each appear on a line by itself. Each must begin in Area A and be followed by a period and a space.

10.6 COMMENT LINES

A comment line is any line with an asterisk in the Continuation Area of the line. A comment line can appear as any line in a source program excluding the first and last lines. Any combination of the characters from the computer's character set may be included in Area A and Area B of that line. The asterisk and the characters in Area A and Area B will be produced on the listing but serve as documentation only.

The continuation of comment lines in the sense of 10.2.2 is not permitted, but successive comment lines are allowed.

CHAPTER 11

RESERVED WORDS

The following is a list of reserved words.

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BEGINNING
BLANK
BLOCK
BY
CALL
CANCEL
CF
CH
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMP
COMP-n
COMPUTATIONAL
COMPUTATIONAL-n
COMPUTE
CONFIGURATION
CONSTANT
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
CURRENCY
DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DISPLAY
DISPLAY-n
DIVIDE
DIVISION
DOWN
ELSE
END
END-OF-PAGE
ENDING
ENTER
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXCEEDS
EXIT
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
HOLD
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEX-n
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
KEYS
LABEL
LAST
LEADING
LEFT
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS
MEMORY
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
OPEN
OPTIONAL
OR
OUTPUT
OV
OVERFLOW
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POSITION
POSITIVE
PREPARED
PRIORITY
PROCEDURE
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID
QUOTE
QUOTES
RANDOM
RANGE
RD
READ
RECORD
RECORDING
RECORDS
REDEFINES

REEL
RELEASE
REMAINDER
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
RF
RH
RIGHT
ROUNDED
RUN
SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SELECTED
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
SUM
SUPERVISOR
SUSPEND
SYNC
SYNCHRONIZED
TALLY
TALLYING
TAPE
TERMINATE
THAN
THROUGH
THRU
TIMES
TO
TYPE
UNEQUAL
UNIT
UNTIL
UP
UPON
UPPER-BOUND
UPPER-BOUNDS
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

Announcement of New NBS Handbook on CODASYL COBOL

Superintendent of Documents,
Government Printing Office,
Washington, D.C. 20402

Dear Sir:

Please add my name to the announcement list for any changes to this document or new additions to CODASYL COBOL.

Name ....

Company ....

Address ....

City .... State .... Zip Code ....

(Notification key N-383)

☆ U. S. GOVERNMENT PRINTING OFFICE : 1969 O - 354-437